GUIA DEL USUARIO

dBase III y dBase II

George Burns



ra-ma

GUIA DEL USUARIO

dBase III y dBase II

George Burns



Título de la obra original:
UNDERSTANDING dBASE III and dBASE II
Copyright (c) George Burns
Publicado en el Reino Unido por: SIGMA PRESS

Copyright de la edición española: RA-MA 1987
Traducción y adaptación: M.C. Dopazo

Editado por:
RA-MA Editorial
Ctra. de Canillas, 144
28043-MADRID

I.S.B.N. 84-86381-24-X
Depósito Legal: M- 1290-1987
Impreso en España
Signo Impresores, S.A.- Madrid

PREFACIO

dBaseII/III es un paquete de software producido por Ashton-Tate Ltd Es un sistema de manejo de bases de datos relacionales. Cualquier software de manejo de bases de datos está diseñado para ayudar al usuario a organizar sus datos de forma que le permita la mayor flexibilidad. El uso de cualquier programa se incrementa en gran medida entendiendo cómo realiza las tareas y conociendo los principales programas del sistema operativo.

Los primeros capítulos tratan el lenguaje de comandos común a ambos, dBaseII y dBaseIII. Se describen los comandos individuales y, si el uso en dBaseIII difiere, se explican los cambios. Se describe el formateo de pantallas usando las herramientas y comandos de pantalla y se da gran cantidad de ejemplos. Se describe el dFormat, herramienta de formateo de pantallas suministrada con el dBaseIII, ya que difiere del ZIP, el programa de formateo de pantallas suministrado con dBaseII. Se incluyen gran cantidad de ejemplos de programas completos, se usan como ejemplos utilizando diversas técnicas, listos para la práctica día a día. Finalmente, se explican las diferencias entre dBaseII y dBaseIII y sus implicaciones para la programación en dBaseIII. Se proporcionan apéndices separados para dBaseII y dBaseIII.

George Burns

NOTA DEL TRADUCTOR:
Para la comprobación de los programas y la traducción de este libro se han usado las siguientes versiones del dBase:

dBaseII Versión 2.4 en inglés (no existe en español).

dBaseIII Versión 1.10 en español.

Ambos se han montado sobre un PC/XT de IBM, con PC-DOS 3.1 para el dBaseIII y PC-DOS 2.0 para dBaseII.

CONTENIDO

CAPITULO 1

Introducción

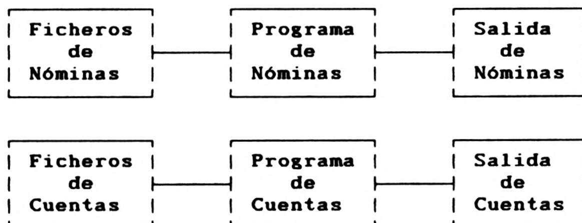
El dBaseII lleva algún tiempo disponible en España. Recientemente ha aparecido el dBaseIII para el IBM PC. Este libro describe el lenguaje de comandos común a ambas versiones del producto y explora algunos de las mejoras que ha introducido el dBaseIII al sistema de manejo de bases de datos del dBase.

Bases de Datos

¿Qué es una base de datos? No está de más que pensemos en ello en este momento, para tener una imagen clara que asiente nuestras ideas. Antes de intentar responder a esta pregunta, debemos considerar qué se entiende por dato. De forma general, los datos pueden aparecer en diversos formtos, ya que se suelen considerar como información. Como ejemplos tenemos los periódicos, facturas, directorios telefónicos y enciclopedias. No importa la forma en que se nos presente la información, debemos leerla y acceder a ella. Hay dos formas diferentes de hacerlo: secuencialmente o aleatoriamente. El acceso secuencial significa comenzar por el principio y leer todos los datos en orden hasta localizar el que nos interesa. El acceso aleatorio significa ir directamente a la parte relacionada con el dato buscado, ej. las páginas de deportes del periódico, para encontrar el dato que nos interesa.

Una base de datos es un método de organización de los datos, como nombres y direcciones, alumnos y notas de exámenes, de una forma estándar para permitir el acceso a los datos. Este acceso es controlado por un sistema de gestión de base de datos, DBMS para abreviar (Data Base Management System). Un DBMS actúa como una conexión entre el usuario y los datos, facilitando a los programas individuales de aplicación el acceso a todos los datos. Aunque los datos están almacenados en ficheros de base de datos, se puede acceder a ellos por medio del DBMS; mientras que un sistema de ficheros requiere programas nuevos de aplicación para cada tarea, cada una con un nuevo juego de ficheros.

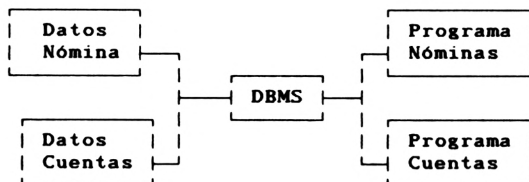
Un sistema de manejo de ficheros se suele configurar así:



El programa de nóminas procesa los ficheros de nóminas. El programa de cuentas procesa los ficheros de cuentas. Para obtener informes que combinen datos de diferentes ficheros se debe escribir un nuevo programa. Los problemas que se pueden encontrar al hacerlo suelen ser, incompatibilidad de datos de un fichero a otro y/o datos que pueden estar ocultos tan profundamente dentro de otros programas, que obtenerlos puede resultar más problemático que beneficioso.

Sistema de Gestión de Bases de Datos

Un Sistema de Gestión de Bases de Datos organiza los datos y hace que sea más fácil obtener información de sus registros. Conceptualmente un DBMS (sistema de gestión de bases de datos) es algo así:



Los datos son dirigidos por el DBMS, y no por los programas individuales de aplicación. Todos los programas de aplicación tienen acceso a todos los datos; en un sistema de manejo de ficheros, esto requeriría una gran cantidad de datos duplicadosy, aparte de los errores potenciales en la introducción de los datos, la integridad de los datos sería muy difícil de mantener.

La forma en que almacena la información el dBaseII/III es muy parecida a series de carpetas en una estantería. Cada carpeta puede tener un tamaño diferente y contiene tipos de información diferentes, ej. carácter, numérica o lógica. Las carpetas en una fila horizontal se llaman registros y contienen información de una entrada específica. Las siguientes filas representan entradas o registros sucesivos. Las carpetas individuales dentro de un registro se llaman campos. Una base de datos que use este tipo de estructura se conoce como base de datos relacional. La estructura de los registros es fijada por la base de datos, aunque diferentes bases de datos pueden tener estructuras distintas. He aquí unos ejemplos:

ESTRUCTURA DEL REGISTRO DE NOMBRE Y DIRECCION

CAMPO	NOMBRE	TAMAÑO	TIPO
1	NOMBRE	25	C
2	APELLIDO1	25	C
3	CALLE	30	C
4	CIUDAD	30	C
	etc.		

La ventaja de este tipo de estructura sobre un sistema normal de ficheros, reside en el lenguaje de acceso de la base de datos y de las facilidades de informes. Un lenguaje de acceso permite al usuario

ESTRUCTURA DEL REGISTRO DE CALIFICACIONES DE EXAMENES

CAMPO	NOMBRE	TAMAÑO	TIPO
1	INGLES	3	N
2	FRANCES	3	N
3	MATEMATICAS	3	N
4	FISICA	3	N
5	QUIMICA	3	N

hacer preguntas relacionadas con la información contenida en la base de datos. Por ejemplo, a la base de datos de nombres y direcciones se la puede pedir una lista de todas las personas que viven en una ciudad concreta.

	CAMPO1	CAMPO2	CAMPO4	CAMPO5
REGISTRO1				
REGISTRO2				
REGISTRO3				
REGISTRO4				
REGISTRO5				
REGISTRO6				
REGISTRO7				
REGISTRO8				
REGISTRO9				
REGISTRO10				
REGISTRO11				

Figura 1.1 - Base de Datos Relacional

En la Figura 1.1 se muestra la imagen de una base de datos relacional.

```
.DISPLAY ALL FOR CIUDAD = "VALENCIA"
```

es un ejemplo de un comando de dBaseIII para listar por la pantalla todos los registros en los que el campo CIUDAD sea Valencia.

En la base de datos de calificaciones de examen, se requiere una lista de los candidatos que hayan pasado la prueba de Inglés:

```
.DISPLAY ALL FOR INGLES >50          (Asumimos que 50 es la nota mínima)
```

Aunque las preguntas individuales sean útiles, sería una ventaja

poder obtener un informe impreso de los datos. dBaseIII nos permite hacerlo con el comando REPORT. Una facilidad adicional es la posibilidad de reunir comandos individuales dentro de programas que realizan funciones rutinarias.

Una base de datos es una colección de información organizada de una forma particular, que permite a un programa de gestión de la base de datos, acceder a esta información. dBaseII/III es un ejemplo de base de datos relacional que soporta comandos de una sola línea, o comandos controlados por programa, e informes. Las principales operaciones en un sistema de gestión de bases de datos son:

1) Crear la estructura de la base de datos;
2) Introducir los datos dentro de la base de datos;
3) Recuperar y añadir datos;
4) Obtener informes;
5) Conectarse con otros programas.

Las cuatro primeras opciones son las tradicionales. Sin embargo, la opción 5 se gestiona por medio de la facultad del dBaseII/III de crear ficheros de datos con diferentes formatos, ej. El BASIC de Microsoft usa comas entre los campos de datos, dBaseIII puede crear ficheros de datos con separadores de comas.

El Lenguaje de Comandos del dBaseII/III

El lenguaje de comandos del dBaseII/III puede realizar cuatro tareas distintas:

1) Crear una estructura de una base de datos;
2) Introducir y recuperar datos de una base de datos;
3) Manipular datos de una base de datos a otra o dentro de una misma base de datos;
4) Producir informes de una base de datos dada.

En este, y en otros capítulos sobre lenguaje de comandos, se examinan los comandos apropiados para cada una de las operaciones anteriores. Se da una idea general con instrucciones que hacen uso de los comandos adecuados. A continuación se explican los comandos con su formato de uso. Si el formato difiere en el dBaseIII, se tratará por separado.

Crear una Base de datos

Ya se ha descrito anteriormente el formato general de una base de datos relacional. Para crear una base de datos en dBaseII/III debemos proporcionar suficiente información para describir cada campo. dBaseII/III requiere lo siguiente:

NOMBRE DE CAMPO
TIPO DE CAMPO
TAMAÑO DEL CAMPO
DECIMAL (número de posiciones después del punto, en un campo numérico)

NOMBRE DE CAMPO
El nombre de campo puede tener hasta 10 caracteres de longitud y debe comenzar con una letra. No se permiten espacios ni comas, y los dos puntos(_ en dBaseIII), si se usan, no deben ser el último carácter, ej.

A	VALIDO
A1234	VALIDO
JOB:número	VALIDO
A123,456	INVALIDO (coma dentro del nombre)
A123:	INVALIDO (dos puntos al final)

TIPO DE CAMPO
Un campo puede ser numérico, alfa-numérico o lógico. El tipo de cada campo se indica mediante la respuesta:

N	numérico;
C	carácter;
L	lógico.

En dBaseIII hay otros dos tipos de campo:

D	fecha en formato DD/MM/AA, ó MM/DD/AA en formato americano;
M	MEMO, texto de hasta 4000 caracteres

TAMAÑO DE CAMPO
Si el campo es numérico o de carácter, puede tener una longitud de hasta 254 caracteres en dBaseII y hasta 4000 en dBaseIII. Tenga en cuanta que en un campo numérico con valor decimal, el punto decimal ocupa una posición de carácter, ej. para un campo numérico con un rango desde 1.00 a 100.00 el tamaño del campo será de 6 (5 dígitos + 1 para el punto decimal).

Para crear un fichero de base de datos se debe diseñar la estructura sobre papel. Consideremos los siguientes ejemplos:

1) Una base de datos de nombres y direcciones de proveedores;
2) Una base de datos de tipos de productos e identificación del proveedor;
3) Una base de datos de pedidos.

Estas tres bases de datos se usarán para ilustrar la variedad de comandos y las operaciones disponibles en el lenguaje dBaseII/III.

La base de datos (1), nombre y dirección de proveedor, puede tener la siguiente estructura:

Nombre proveedor	NOM:PROV	C	45
Dirección proveedor	DIR:PROV	C	45
Teléfono proveedor	TEL:PROV	C	12
Contacto proveedor	CONT:PROV	C	30
Identificación proveedor	ID:PROV	N	3

La única dificultad que puede presentar esta estructura está en el campo de la dirección. La mejor solución es dividir este campo de la siguiente forma:

CALL:PROV	C	25
CIUD:PROV	C	25
REG:PROV	C	25

Para crear este fichero, cargue el dBase.

```
A>DBASE   (asumimos que el dBaseII/III está en el dispositivo A)
.CREATE
ENTER FILENAME: PROVEED
ENTER RECORD STRUCTURE AS FOLLOWS:
 FIELD   NAME,TYPE,WIDTH,DECIMAL PLACES
 001     NOM:PROV,C,45
 002     CALL:PROV,C,25
 003     CIUD:PROV,C,25
 004     REG:PROV,C,25
 005     TEL:PROV,C,12
 006     CONT:PROV,C,30
 007     ID:PROV,N,3
```

El proceso se repite hasta que se ha introducido la estructura de todos los campos. Después del último campo, pulse la tecla RETURN.

En dBaseIII el procedimiento no ha cambiado, pero la estructura de la pantalla es diferente:

```
B:PROVEED.DBF                                 Bytes restantes:   3835
                                              Campos definidos:     7

   Nomb campo  Tipo       Ancho  Dec          Nomb campo  Tipo      Ancho  Dec
   -------------------------------            -------------------------------
 1 NOM_PROV    Car/texto    45
 2 CALL_PROV   Car/texto    25
 3 CIUD_PROV   Car/texto    25
 4 REG_PROV    Car/texto    25
 5 TEL_PROV    Car/texto    12
 6 CONT_PROV   Car/texto    30
 7 ID_PROV     Numérico      3    0
```

El cursor se coloca en la primera línea, donde se requiere el nombre del campo. El valor que toma el campo por omisión es carácter. Si es el correcto, pulse simplemente RETURN; si no lo es, introduzca una de las siguientes opciones:

L para lógico
N para numérico
D para fecha
M para MEMO

Los campos FECHA(D) y LOGICO(L) se ponen automáticamente con longitudes 8 y 1 respectivamente. Si se selecciona el tipo NUMERICO (N), pide el número de cifras decimales;, si no, se salta. Cuando se han preparado todos los campos, se salva la estructura tecleando ^END.

El dBaseII/III nos preguntará ahora si queremos introducir datos. Teclee S(Y) y, si tiene edición de pantalla completa, verá un registro en blanco en la pantalla.

```
RECORD # 00001
NOM:PROV      :                                             :     (45 ESPACIOS)
CALL:PROV     :                         :                         (25 ESPACIOS)
CIUD:PROV     :                         :                         (25 ESPACIOS)
REG:PROV      :                         :                         (25 ESPACIOS)
TEL:PROV      :            :                                      (12 ESPACIOS)
CONT:PROV     :                              :                    (30 ESPACIOS)
ID:PROV       :   :                                                (3 ESPACIOS)
```

Ahora puede introducir los datos dentro de cada campo; después de último, pulse RETURN para continuar introduciendo. Si comete un error mientras introduce los datos, hay varias posibilidades de corregirlo. Si está todavía dentro del campo, use la tecla de retroceso para volver al error y corregirlo. Si ya ha pasado al campo siguiente, use CTRL E (Tecla Ctrl junto con la E) para volver al campo anterior. Repitiendolo puede moverse un campo hacia atrás cada vez.

El campo MEMO en dBaseIII se usa para almacenar texto. La entrada de datos dentro de estos campos se hace a través de un procesador de textos que viene incorporado en el paquete. Cuando tenga el cursor sobre el campo MEMO, pulse ^PgUp (CTRL y la tecla PgUp al mismo tiempo). Con esto haremos que se active el procesador de textos. Una vez activo, se puede teclear el texto directamente que se justificará automáticamente a la derecha. Tiene disponibles los siguientes comandos de edición:

Tecla	Función
^PgUp	Entrar en el procesador de textos desde el campo MEMO;
Flechas	Mueve el cursor en la dirección adecuada;
Tabulador	Mueve el cursor al siguiente tabulador;
PgUp	Mueve hacia arriba un página de texto;
PgDn	Mueve hacia abajo una página de texto;
DEL	Borra el carácter sobre el que está el cursor;
^V	Bascula el modo inserción activo/inactivo
^END	Termina y salva el texto.

Si se van a introducir datos posteriormente, se deben ejecutar los siguientes pasos.

1) Decirle al dBaseII/III la base de datos que se va a usar;
2) Decirle al dBaseII/III que añada un registro en blanco.

```
.USE PROVEED          * Le dice al dBaseII que abra el fichero PROVEED.DBF

.APPEND               * añade un registro en blanco y permite que se introduzcan datos en ese
                        registro

RECORD # 00005
NOM:PROV      :                                             :     (45 ESPACIOS)
CALL:PROV     :                         :                         (25 ESPACIOS)
CIUD:PROV     :                         :                         (25 ESPACIOS)
REG:PROV      :                         :                         (25 ESPACIOS)
TEL:PROV      :            :                                      (12 ESPACIOS)
CONT:PROV     :                              :                    (30 ESPACIOS)
ID:PROV       :   :                                                (3 ESPACIOS)
```

En dBaseIII la pantalla para añadir registros es similar a la anteriores, salvo que aparecen unas líneas de ayuda del editor en la parte superior de la pantalla, similares a las que aparecen en modo ASSIST:

```
| CURSOR  <-- -->   |          ARR   ABJO  |  ELIMINAR      | Modo insertar: Ins |
| Car:    <-- -->   | Campo:    ↑     ↓    | Car:       Del | Salir:        ^End |
| Pal:    Home End  | Pag:     PgUp  PgDn  | Campo:      ^Y | Cancelar:      Esc |
|                   | Ayuda:     F1        | Registro:   ^U | Memo:       ^Pg Dn |
```

Este método de añadir registros se puede repetir siempre que no se tengan que añadir muchos. Si se van a añadir más de cinco registros, este método puede ser tedioso. dBaseII/III permite agrupar los comandos y ejecutarlos como un programa, con lo que se consigue que la entrada de datos sea más fluida (ver más adelante).

Para completar las bases de datos que se van a usar, se requieren las siguientes estructuras:

PRODUCTO

NOMBRE CAMPO	TIPO	ANCHO	DECIMAL
TIPO:PROD	C	30	
COSTO:PROD	N	6	
ID:PROV	N	3	
NOMBRE:PROD	C	30	

PEDIDOS

NOMBRE CAMPO	TIPO	ANCHO	DECIMAL
NOM:PROV	C	35	
CALL:PROV	C	35	
CIUD:PROV	C	35	
ITEM1	C	40	
ITEM2	C	40	
ITEM3	C	40	
ITEM4	C	40	
COST1	N	6	2
COST2	N	6	2
COST3	N	6	2
NUM1	N	3	
NUM2	N	3	
NUM3	N	3	
TOTAL	N	6	

Cree ahora estas estructuras, pero no introduzca ningún dato por el momento.

Las tres estructuras de base de datos que hemos preparado están ahora presentes en los ficheros que aparecerán en el diskette o en el disco duro como:

PROVEED.DBF
PEDIDOS.DBF
PRODUCTO.DBF

La extensión DBF indica que los ficheros son bases de datos. Puede suceder que haya omitido un campo o que la estructura de la base de datos se haga demasiado pequeña. Para corregir estos errores se debe modificar la estructura.

Como ejemplo, supongamos que queremos cambiar el tamaño del campo de contacto del proveedor, a 35 caracteres. Debemos usar los siguientes comandos en este orden:

```
.USE PROVEED                  * Selecciona el fichero que hay que abrir
.COPY TO TEMP                 * Copia el contenido de PROVEED a un fichero llamado TEMP
.MODIFY STRUCTURE             * Le dice al dBaseII que se va a cambiar la estructura
                                de PROVEED.
```

Aparecerá un mensaje en la pantalla advirtiéndole que se borrarán todos los datos de la base de datos, y le dará la opción de seguir o salir. El uso del comando COPY nos proporciona una copia de PROVEED en TEMP. Teclee 'Y' para hacer las modificaciones. La pantalla mostrará ahora la estructura de la base de datos PROVEED:

	NAME	TYP	LEN	DEC	
FIELD 01	:NOM:PROV	C	045	000	:
FIELD 02	:CALL:PROV	C	025	000	:
FIELD 03	:CIUD:PROV	C	025	000	:
FIELD 04	:REG:PROV	C	025	000	:
FIELD 05	:TEL:PROV	C	012	000	:
FIELD 06	:CONT:PROV	C	030	000	:
FIELD 07	:ID:PROV	N	003	000	:

Para cambiar el tamaño del campo CONT:PROV, sitúe el cursor sobre esa entrada, pase al campo de longitud (LEN) e introduzca el cambio. Para salir de la secuencia de modificación, hay dos opciones disponibles:

1) CTRL W — **(Tecla Ctrl y W al mismo tiempo) Esta opción escribe los cambios en el disco.**
2) CTRL Q — **(Tecla Ctrl y Q al mismo tiempo) Esta opción interrumpe la secuencia de modificación sin salvar los cambios.**

Una vez modificado el tamaño del campo CONT:PROV, copie los datos desde TEMP.DBF a la base de datos PROVEED.DBF. Para esta operación se usa la siguiente serie de comandos:

```
.USE PROVEED           * Identifica la base de datos que vamos a usar
.APPEND FROM TEMP      * Escribe los datos de TEMP en PROVEED
.DELETE FILE TEMP      * borra el fichero TEMP del disco, si no se necesita más.
.QUIT
```

El procedimiento para añadir registros en dBaseIII es diferente. En dBaseII es necesario copiar la base de datos a un fichero temporal.

antes de modificar realmente la estructura. El comando MODIFY de dBaseIII hace la copia automáticamente y entra después en la secuencia de modificación. El mensaje de aviso que aparece en dBaseII no aparece. En modo ASSIST los comandos de edición aparecen en la pantalla:

CURSOR <-- -->	ARR ABJO	ELIMINAR	Modo insertar: Ins
Car: <-- -->	Campo: ↑ ↓	Car: Del	Salir: ^End
Pal: Home End	Pag: PgUp PgDn	Campo: ^Y	Cancelar: Esc
	Ayuda: F1	Registro: ^U	Memo: ^Pg Dn

Debe tener en cuenta que si se reduce el tamaño de un campo, los datos que se recuperan en ese campo pueden aparecer truncados para acomodarlos a la nueva longitud.

Si se ha de cambiar el nombre de un campo, se debe seguir un procedimiento diferente. Cuando se copian los datos a un fichero temporal, estos se almacenan de la misma forma que en el original, por nombre de campo. Si se requiere cambiar un nombre de campo, los datos en la base de datos se deben almacenar por posición, en lugar de por nombre. Para almacenar los datos por posición se debe usar un comando SDF:

```
.USE PROVEED
.COPY TO TEMP.TXT SDF                  * Copiar a TEMP.TXT los datos solamente
.MODIFY STRUCTURE
.APPEND FROM TEMP.TXT SDF              * añadir los datos desde TEMP.TXT
```

NOTA. La extensión TXT indica que solamente se almacenan los datos. Este procedimiento se debe usar solamente para cambiar los nombres de los campos, ya que en el fichero SDF se almacenan solamente por posición. Cualquier cambio en el tamaño de los campos destruye la relación de la posición.

En dBaseIII, los cambios en el nombre y tamaño de los campos se realizan en un proceso de dos pasos:

1) Cambia primero el nombre del campo
2) Cambia el tamaño del campo

En cada paso de este proceso, el dBaseIII hace una copia de seguridad de la base de datos.

Las estructuras de base de datos que hemos preparado anteriormente se han introducido desde el teclado. Si en una aplicación particular se usa un fichero solamente durante una parte de este tiempo, puede ser más fácil crear el fichero durante la aplicación. Para conseguirlo se puede crear una base de datos que contenga la información para crear este fichero temporal. La secuencia de operaciones es:

1) Crear de un fichero desde el teclado
2) salvar la estructura del fichero
3) ahora se puede crear un fichero desde el fichero de estructura

Por ejemplo, supongamos que necesitamos fichero temporal que contiene los siguientes campos:

NOM:PROV
CONT:PROV
TELE:PROV

A este fichero le llamaremos TPROV y lo crearemos como antes:

```
.CREATE TPROV
```

NOTA. Si introducimos el nombre del fichero en la misma línea del comando CREATE, pasamos directamente a la parte en que se define la estructura de los campos, tanto en dBaseII como en dBaseIII.

CAMPO	**NOMBRE**	**TIPO**	**ANCHO**	**DECIMAL**
001	NOM:PROV	C	35	
002	CONT:PROV	C	35	
003	TELE:PROV	C	12	

En dBaseIII la pantalla será:

```
B:TPROV.DBF                                   Bytes restantes:   3918
                                              Campos definidos:     3

   Nomb campo  Tipo      Ancho  Dec          Nomb campo  Tipo      Ancho  Dec
   ------------------------------            ------------------------------
 1 NOM_PROV    Car/texto   35
 2 CONT_PROV   Car/texto   35
 3 TEL_PROV    Car/texto   12
```

Ya que solo necesitamos la estructura de este fichero, no necesitamos introducir datos. Cuando aparezca el punto, introduzca la siguiente secuencia de comandos:

```
.USE TPROV
.COPY TO TPROV1 STRUCTURE EXTENDED
```

El fichero TPROV1.DBF contiene ahora la información sobre la estructura del fichero TPROV.DBF. TPROV1.DBF contendrá un registro que describe cada campo del fichero TPROV.DBF. TPROV.DBF contiene tres campos, por lo tanto, TPROV1.DBF contendrá tres registros:

REGISTRO	**CAMPO1**	**CAMPO2**	**CAMPO3**
00001	NOM_PROV	C	35
00002	CONT_PROV	C	35
00003	TELE_PROV	C	12

Cada registro de TPROV1.DBF contiene toda la información relacionada con un campo del fichero TPROV.DBF. Para crear un fichero usando esta estructura, el comando debe ser:

```
.CREATE TPROV FROM TPROV1
```

Si necesita crear ficheros en una unidad diferente de la que tiene

por defecto, el comando se debe modificar a:

```
.CREATE B:TPROV FROM TPROV1
```

o

```
.CREATE TPROV FROM B:TPROV1
```

En el primer ejemplo, el fichero TPROV se creará en la unidad B, desde el fichero de estructura de la unidad A (suponiendo que A sea la unidad por defecto). En el segundo ejemplo, el fichero TPROV se creará en la unidad por defecto desde el fichero de estructura que hay en la unidad B.

Las operaciones básicas de creación y uso de bases de datos se pueden realizar usando los siguientes comandos:

CREATE
El comando CREATE se usa para preparar una nueva base de datos. Puede tener un nombre de fichero o puede usar un fichero de estructura. Se pueden referir unidades distintas de la que se asume por defecto.

a) CREATE — Inicia la secuencia de creación del dBaseII

b) CREATE fichero — Hace que la secuencia de creación omita la petición de nombre de fichero.

c) CREATE B:fichero — Crea un fichero en la unidad B o en cualquier otra unidad permitida que se especifique.

d) CREATE fichero FROM estructura
Crea un fichero desde un fichero de estructura

MODIFY STRUCTURE
Permite alterar los tamaños de los campos y/o sus nombres. Se debe tener mucho cuidado cuando se alteran los nombres de los campos.

APPEND
Se puede usar para añadir un registro en blanco a la base de datos o para añadir registros desde otra base de datos. Por ejemplo:

```
.APPEND FROM TEMP
```

añadirá registros desde el fichero TEMP al fichero abierto con el comando USE que se ha introducido más recientemente.

DELETE FILE
Se usa para borrar un fichero del disco.

```
.DELETE FILE TEMP

.DELETE FILE TEMP.DBF     en dBaseIII
```

borrará el fichero TEMP.DBF, si no está abierto. Para el dBaseIII es necesario poner la extensión DBF.

USE
Selecciona la base de datos que se va a abrir para ser usada. Si se ha introducido un comando USE previamente, el fichero abierto por ese comando se cerrará y se abrirá el que se ha especificado en el último.

```
.USE NOMBRES                * abre el fichero NOMBRES.DBF
.APPEND                     * añade un registro en blanco
.USE PROVEED                * cierra el fichero NOMBRES y abre el fichero PROVEED

.COPY TO TEMP               * copia a todos los registros de PROVEED a TEMP
.USE TEMP                   * cierra PROVEED y abre TEMP
.DELETE FILE TEMP           * mensaje de error, intenta borrar un fichero abierto,
                              que es el fichero actualmente abierto
.USE                        * cierra todos los ficheros abiertos
.DELETE FILE TEMP.DBF       * esta vez no da error
  o
.DELETE FILE TEMP           * en dBaseII
```

Recuerde añadir la extensión DBF si está usando dBaseIII.

Los comandos CREATE y APPEND permiten al usuario preparar e introducir datos dentro de una base de datos. Es posible examinar esos datos en la pantalla usando comandos simples desde el teclado. Los siguientes comandos mostrarán en la pantalla los registros de la base de datos seleccionada:

```
.USE NOMBRES
.DISPLAY
```

La pantalla nos mostrará el primer registros del fichero NOMBRES. Los datos se escriben en la pantalla por campos. Si el tamaño de los registros excede los 80 caracteres, el registro se dividirá en dos líneas y será difícil leerlo. Los números de los registros se pueden suprimir añadiendo OFF al comando DISPLAY, de la siguiente forma:

```
.USE NOMBRES
.DISPLAY OFF
```

Veremos en la pantalla el primer registro de la base de datos, pero esta vez sin el número del registro precediéndolo. Para mostrar un grupo de registros al mismo tiempo, se modifica el comando DISPLAY de la siguiente forma:

```
.USE NOMBRES                * use el fichero NOMBRES
.DISPLAY NEXT 5             * muestra los siguientes cinco registros con número de registro
.DISPLAY RECORD 10          * muestra un registro seleccionado
.DISPLAY RECORD 10 OFF      * Muestra el registro 10 sin número de registro
.GO TOP                     * se sitúa en el primer registro de la base de datos
.? RECNO()                  * imprime el número del registro
```

en dBaseII

```
.? #                        * # función de número de registro en dBaseII
```

El símbolo de impresión en pantalla de dBaseII/III es ?. En dBaseII la función que nos da el número de registro se llama mediante #. En dBaseIII, el número del registro se obtiene mediante la función

RECNO(). La impresión se situará en la línea siguiente al ?. Si se requiere imprimir sobre la misma línea, en lugar de ? se usa ??.

```
.GO BOTTOM                  * ir al último registro
.? RECNO()                  * imprimir el número del registro
.GO TOP
.SKIP 5                     * ir al quinto registro
.? RECNO()                  * imprimir el número del registro
.SKIP -2                    * volver atrás dos registros
.? RECNO()                  * deberá imprimir 3
```

NOTA. En dBASEII la función RECNO() se debe sustituir por el símbolo #.

Las funciones SKIP, GO TOP, GO BOTTOM, ? y RECNO() son muy útiles cuando examinamos los registros con comandos desde el teclado. Dentro de una base de datos que contiene muchos registros, es muy útil seleccionar mediante un criterio de selección. El comando DISPLAY se puede modificar para obtener esta facilidad.

```
.USE NOMBRES
.DISPLAY ALL FOR APELL='PEREZ'
```

Esto nos mostrará en la pantalla todas las entradas en las que el campo del apellido contenga PEREZ, de 15 en 15 registros. Si se usa una cantidad numérica para la selección, se puede usar el siguiente comando:

```
.USE PRODUCTO
.DISPLAY ALL FOR COSTO_PROD = 100.00
```

Este comando seleccionará todos los registros en los que el costo del producto sea 100.00, y los mostrará de 15 en 15.

El comando DISPLAY se puede usar para obtener información del sistema:

```
.USE PRODUCTO
.DISPLAY STRUCTURE
```

En la pantalla aparecerá la estructura del fichero PRODUCTO.DBF. Con este comando se puede obtener la información de la estructura de cualquier fichero. Se puede especificar una unidad alternativa:

```
.USE B:PRODUCTO
.DISPLAY STRUCTURE
```

El comando DISPLAY STRUCTURE muestra la estructura del fichero actualmente en uso.

Cuando se desarrollan programas, es muy útil poder comprobar qué ficheros están presentes en las unidades del sistema. Para esto se puede usar el comando DISPLAY:

```
.DISPLAY FILES LIKE *.COM
```

que nos mostrará todos los ficheros que tengan la extensión COM (CP/M) en la unidad actual. Se puede obtener una lista de los ficheros contenidos en otra unidad con:

```
.DISPLAY FILES ON B LIKE *.PRG
```

que mostrará los ficheros que tengan la extensión PRG en la unidad B o en cualquier otra unidad del sistema.

Una alternativa al comando DISPLAY es LIST. La diferencia principal entre DISPLAY y LIST es que el segundo muestra todos los registros en la pantalla sin parar. Si tiene una impresora conectada a su ordenador, introduzca el siguiente comando:

```
.USE NOMBRES
.SET PRINT ON
.LIST ALL FOR APELL = 'PEREZ'
.SET PRINT OFF
```

Obtendremos en la impresora el listado de los registros cuyo campo APELL sea igual a PEREZ, al mismo tiempo que en la pantalla. El comando LIST funciona de la misma forma que DISPLAY, con la excepción de que no parará cada 15 registros. El comando SET PRINT ON hace que todo lo que salga a pantalla se mande también a la impresora, mientras que SET PRINT OFF elimina esta impresión. Estos dos comandos tienen un efecto similar a Ctrl P en CP/M y Alt P en MS/DOS. El formato general de los comandos DISPLAY y LIST es:

```
        [ALL     ]
DISPLAY [num-reg] [OFF] FOR [condición]
        [NEXT n  ]

LIST [OFF] FOR [condición]
```

Si se usa LIST para buscar en una base de datos, se puede detener el movimiento de los registros en la pantalla, con Ctrl S, y reanudarlo, con Ctrl S otra vez. En dBaseIII aparecen el número del registro y los nombres de los campos.

```
Reg. Nº   NOMBRE_CAMPO1  NOMBRE_CAMPO2 NOMBRE_CAMPO3
```

Una facilidad adicional de LIST y DISPLAY en dBaseIII es mostrar la información del estado del sistema. Su formato es:

```
.DISPLAY STATUS [TO PRINT]

.LIST STATUS [TO PRINT]
```

Ambos comandos proporcionan la misma información, pero con LIST no se detiene la pantalla cuando se llena, mientras que DISPLAY espera a que se pulse una tecla para avanzar una vez que se ha llenado la pantalla.

La información que se obtiene es:

Nombre de la base de datos en uso
Número del área de trabajo
Seudónimo (Alias)
Nombre del fichero índice abierto
Relaciones de la base de datos
Clave de índice de los ficheros índice abiertos

Este comando incluye también esta otra información:

Camino actual de búsqueda de ficheros
Unidad de discos por defecto
Posición de los comandos SET ON/OFF
Especificación de margen izquierdo
Asignación de las teclas de función

Ambos, LIST y DISPLAY, proporcionan todos los campos de todos los registros que coincidan con la condición de selección. Generalmente es mejor seleccionar los campos requeridos, ya que cuando es grande el número de campos, el resultado será confuso en la pantalla. Aunque LIST y DISPLAY permiten especificar una lista de campos, nos será de más utilidad el comando BROWSE.

BROWSE

El comando BROWSE se puede usar de dos formas. BROWSE por sí mismo mostrará hasta 19 registros y tantos campos como quepan en la pantalla. Si se le suministra una lista de campos, solo se listarán esos campos.

```
.USE NOMBRES
.BROWSE

Reg. Nø          1    NOMBRES
NOMBRE----------- APELLIDO------- DIRECCION------ TELEFONO----
JAIME             FERNANDEZ       SERRANO 125     91-2-314-056
JORGE             HERNANDEZ       GRAN VIA 27     91-3-714-526
ANDRES            RAMIREZ         VILLALAR 35     91-6-145-672
```

Se muestran los cuatro campos porque su longitud combinada cabe en la pantalla; mediante teclas de control se podrían ver los otros campos que no caben en la pantalla.

```
.USE PROVEED
.BROWSE FIELDS NOM_PROV,CONT_PROV,TEL_PROV

Reg. Nø          1    PROVEED
NOM_PROV------------ CONT_PROV---------- TEL_PROV----
HERRERO HERMANOS     SR GOMEZ HUERTAS    91-2-613-456
GRAFICA MARINA       SR MARTIN ARELLANO  91-2-415-649
```

La mayor facilidad del comando BROWSE es la de permitir realizar cambios en los campos. Se pueden realizar los cambios usando los comandos de pantalla completa para mover el cursor a la entrada deseada, . Para salvar los cambios se usa Ctrl W o, si no se requieren los cambios, Ctrl Q. Para mover la presentación de la pantalla a lo largo de los campos hacia la derecha, se usa Ctrl B, y Ctrl Z para moverla hacia la izquierda.

Secuencias de control en dBaseII

MODO BROWSE

CTRL-B mueve la pantalla hacia la DERECHA
CTRL-Z mueve la pantalla hacia la IZQUIERDA

En dBaseIII, el comando BROWSE está acompañado del menú:

CURSOR	<-- -->		ARR	ABJO	ELIMINAR		Modo insertar:	Ins
Car:	<-- -->	Campo:	↑	↓	Car:	Del	Salir:	^End
Pal:	Home End	Pag:	PgUp	PgDn	Campo:	^Y	Cancelar:	Esc
		Ayuda:	F1		Registro:	^U	Memo:	^Pg Dn

Tecleando ^HOME se accede a un menú de opciones que permite las siguientes operaciones:

FINAL — Se sitúa al final del fichero
PRINCIPIO — Se sitúa al principio del fichero
BLOQUEAR — Define el número de campos a la izquierda de la pantalla que permanecen durante la operación de desplazamiento lateral
REG. Nø — Se sitúa en el número de registro seleccionado
SELECCIONAR — Edita un solo campo

Como en dBaseII, los cambios realizados en un campo son introducidos directamente, situándose en el campo que se quiere modificar mediante las teclas de movimiento del cursor y tecleando el nuevo valor. ^END se usa para terminar y salvar los cambios, ESC se usa para salir sin salvar los cambios. Para desplazar el contenido de la pantalla en dBaseIII, use:

^<-- mueve hacia la izquierda
^--> mueve hacia la derecha

NOTA. Se pueden añadir registros usando la flecha hacia abajo en el final del fichero. dBaseIII le preguntará si quiere añadir registros. Si quiere teclee S, si no quiere pulse RETURN para terminar el modo BROWSE. Los campos MEMO no se pueden editar usando el comando BROWSE.

EDIT
El comando EDIT es una alternativa de BROWSE. BROWSE solo permite tener 19 registros en pantalla al mismo tiempo, con los campos que caben en la pantalla solamente. EDIT muestra solamente un registro a un mismo tiempo, pero le permite acceder a todos los campos, suponiendo que su número no exçeda de 23. La función EDIT se usa de la siguiente forma:

```
.USE PROVEED
.EDIT 1                * Editar el registro número 1 de la base de datos PROVEED
```

NOTA Si no se le suministra número de registro, dBaseII da un mensaje de error y le pide el número de registro antes de continuar.

Teclas de Control

EN MODO EDIT

CTRL-U	Pone o quita la marca de BORRADO al registro
CTRL-C	Escribe en disco el registro que hay en pantalla y AVANZA al siguiente registro
CTRL-R	Escribe en disco el registro que hay en pantalla y VUELVE al registro anterior
CTRL-Q	Ignora los cambios en el registro que hay en pantalla y vuelve al dBase
CTRL-W	Escribe todos los cambios en disco y vuelve

La pantalla se limpiará y mostrará el registro 1 de la base de datos PROVEED de la siguiente forma:

```
Reg. Nº     1
NOM_PROV   :_                                    :
CALL_PROV  :                         :
CIUD_PROV  :                         :
REG_PROV   :                         :
TEL_PROV   :            :
CONT_PROV  :                              :
ID_PROV    :   :
```

El cursor se situará sobre al campo 1. Usando las teclas de control del cursor, éste se puede posicionar en cualquiera de los campos y se puede escribir sobre los datos existentes para modificarlos. Cualquier carácter extra que quede después del cambio se debe borrar usando la barra de espaciado. Para salvar los cambios al disco, use CTRL-W, si no quiere salvar los cambios, use CTRL-Q.

En dBaseIII, el comando EDIT actúa sobre el registro actual si no se le especifica un número de registro con el comando. El movimiento del cursor es similar al de otras funciones de pantalla.

1) Las teclas de flechas se usan para mover el cursor sobre la pantalla.
2) PgUp vuelve al registro anterior
3) PgDn avanza al registro siguiente
4) ^END termina y salva los cambios
5) ESC termina sin salvar los cambios

Para editar campos memo, sitúe el cursor en el campo y pulse ^PgDn. Cuando la edición se haya completado, pulse ^PgUp y salve los cambios o ESC para terminar sin salvarlos.

CODIGOS DE MOVIMIENTO DE PANTALLA COMPLETA - TODOS LOS COMANDOS DE dBaseII

CTRL-X	mueve el cursor hacia ABAJO al siguiente campo (también CTRL-F)
CTRL-E	mueve el cursor hacia ARRIBA al siguiente campo (también CTRL-A)
CTRL-D	mueve el cursor un carácter hacia ADELANTE
CTRL-S	mueve el cursor un carácter hacia ATRAS
CTRL-G	borra el carácter sobre el que está el cursor
<DEL>	borra el carácter que hay a la izquierda del cursor
CTRL-Y	borra el campo sobre el que está situado el cursor
CTRL-V	pone o quita el modo INSERCION
CTRL-W	salva cualquier cambio y vuelve al dBase

EN MODO MODIFY

CTRL-T	BORRA la línea actual, mueve hacia arriba las líneas siguientes
CTRL-N	INSERTA una nueva línea en la posición del cursor
CTRL-C	mueve el texto de la pantalla media página hacia arriba
CTRL-W	escribe en disco los cambios y vuelve al dBase
CTRL-Q	ignora los cambios y vuelve al dBase

EN MODO APPEND

<ENTER>	cuando el cursor está al principio del primer campo, termina el modo APPEND
CTRL-W	escribe el registro en disco y se sitúa en el siguiente
CTRL-Q	ignora el registro actual y vuelve al dBase

TECLAS DE CONTROL SI NO ESTA EN PANTALLA COMPLETA

CTRL-P	activa o desactiva la impresora
CTRL-R	repite el comando dBase que se acaba de ejecutar
CTRL-X	borra la línea de comandos sin ejecutarlo
CTRL-H	mover un espacio hacia atrás
CTRL-M	actúa como un retorno de carro

Control del cursor en dBaseIII

TECLA	ALTERNATIVA	FUNCION
↑	^E	Mueve el cursor una línea o un campo hacia arriba
↓	^X	Mueve el cursor una línea o un campo hacia abajo
<-	^D	Mueve el cursor un espacio hacia la izquierda. En los menús, elige la opción de la izquierda.
->	^S	Mueve el cursor un espacio hacia la izquierda. En los menús, elige la opción de la izquierda.
^->	^B	Mueve el contenido de la pantalla un campo hacia la derecha en BROWSE. En MODIFY REPORT, mueve hacia arriba el contenido de la estructura del fichero. En MODIFY COMMAND, mueve el cursor al final de la línea.
^<-	^Z	Mueve el contenido de la pantalla un campo hacia la izquierda en BROWSE. En MODIFY REPORT, mueve hacia abajo el contenido de la estructura del fichero. En MODIFY COMMAND, mueve el cursor al principio de la línea.
<--	RUB	Borra el carácter que hay a la derecha del cursor.
DEL	^G	Borra el carácter sobre el que está situado el cursor.
END	^F	Mueve el cursor una palabra hacia la derecha,
^PgUp	^W	Salva y sale de modo de pantalla completa.
ESC	^Q	Sale sin salvar los cambios (salva todo menos el registro actual en APPEND y BROWSE) y vuelve al dBase.
HOME	^A	Mueve el cursor una palabra hacia la izquierda
^HOME	F1	Activa o desactiva el menú en los comandos de pantalla completa
INS	^V	Activa o desactiva el modo INSERTAR. Cuando está activado, inserta el carácter delante del cursor. Cuando está desactivado, escribe encima del carácter sobre el que está situado el cursor.
^KW		En MODIFY COMMAND, escribe el fichero completo sobre otro fichero.

^KR		En MODIFY COMMAND, lee otro fichero dentro del fichero actual.
^N		Inserta una línea nueva o una definición de campo.
PgUp	^R	Vuelve al anterior registro, pantalla o ventana de 17 registros en modo BROWSE.
PgDn	^C	Pasa al siguiente registro, pantalla o ventana de 17 registros en modo BROWSE.
^PgDn		Entra y edita campos MEMO.
RETURN		Mueve el cursor al siguiente campo. En APPEND, salva y sale si está el cursor sobre el primer carácter de un registro en blanco. En EDIT, salva y sale si el cursor está en el último campo del último registro. En MODIFY COMMAND, inserta una línea si está activado el modo INSERTAR. En los menús, selecciona una opción.
^T		Borra una palabra a la derecha de la posición del cursor.
^U		Marca un registro como BORRADO en BROWSE y EDIT. BORRA una definición de campo en MODIFY REPORT o MODIFY STRUCTURE.
^Y		Borra hasta el final de la línea, o la línea completa, en MODIFY COMMAND.

CAPITULO 2

Organización de las Bases de Datos

En el capítulo 1 se mostraron los dos métodos usados más comúnmente para acceder a datos: secuencial y aleatorio. Para acceder rápidamente a un dato determinado, se requiere un sistema de índices. dBaseII /III permite al usuario crear ficheros índice para las bases de datos con campo de clave de un solo campo o de campos múltiples.

Los sistemas indexados son comunes en muchas aplicaciones; libros, librerías, ficheros de repuestos y sistemas de almacenamiento, son unos pocos ejemplos. Esencialmente, un índice es una forma de proporcionar información adicional para localizar una entrada determinada en el sistema. Un índice en dBaseII/III realiza la misma función; proporciona información adicional que permite localizar los registros en la base de datos. Un fichero índice es una lista de apuntadores a los registros de la base de datos, ordenados por una clave determinada.

INDICE

Para indexar la base de datos PRODUCTO por la identificación de proveedor (ID_PROD), se debe usar la siguiente secuencia de comandos:

```
.USE PRODUCTO
.INDEX ON ID_PROD TO PRODIDX
```

El comando INDEX requiere el nombre del campo por el que se va a indexar y el nombre del fichero de índice que se va a crear. El fichero de índice tendrá la extensión .NDX. Como la función INDEX crea su propia lista, el fichero de índice, no afecta al orden físico de los datos en la base de datos. Para usar un índice junto con la base de datos, se usa el comando:

```
. USE PRODUCTO INDEX PRODIDX
. DISPLAY ALL
Reg. Nø  TIPO_PROD                 COSTO_PROD ID_PROD NOM_PROD
     1  CLAVOS                          0.05       1 CLAVO1
     3  SIERRA                        920.00       2 SIERRA3
     2  MARTILLO                      325.00       3 MARTILLO1
     4  TORNILLO                        3.75       4 TUERCA3
```

Observe que los datos que aparecen en la pantalla no está en orden de registro, sino en el orden del índice. Repita ahora la operación sin el fichero de índice:

```
. USE PRODUCTO
. DISPLAY ALL
Reg. Nº  TIPO_PROD                     COSTO_PROD ID_PROD NOM_PROD
     1   CLAVOS                              0.05       1 CLAVO1
     2   MARTILLO                          325.00       3 MARTILLO1
     3   SIERRA                            920.00       2 SIERRA3
     4   TORNILLO                            3.75       4 TUERCA3
```

La pantalla muestra ahora los datos por orden de número de registro. El efecto de un fichero de índice en la ordenación de una base de datos es particularmente útil para obtener listados impresos por un orden especificado. Por ejemplo, los datos del fichero NOMBRES se pueden imprimir por orden de nombre si la base de datos se ha indexado por el campo de nombre, así:

```
. USE NOMBRES INDEX NOMIDX
. SET PRINT ON
. LIST
. SET PRINT OFF
```

Para usar un fichero de índice para encontrar un registro concreto, se usa el comando FIND. Con este comando no es necesario identificar el tipo de datos; por ejemplo, comillas para caracteres. El índice ya tiene esta información. En la base de datos de productos, se usan varios códigos de identificación. Para localizar un proveedor cuyo código de identificación sea 1, use los siguientes comandos:

```
. USE PRODUCTO INDEX PRODIDX
. STORE 1 TO SID
. FIND SID
. DISPLAY

Reg. Nº  TIPO_PROD                     COSTO_PROD ID_PROD NOM_PROD
     1   CLAVOS                              0.05       1 CLAVO1
```

NOTA si hay más de un registro con la misma clave, el comando FIND se detiene en la primera ocurrencia que se encuentre.

Ahora introduzca los siguientes comandos:

```
. SKIP 1
. DISP

Reg. Nº  TIPO_PROD                     COSTO_PROD ID_PROD NOM_PROD
     2   MARTILLO                          325.00       3 MARTILLO1

. SKIP -1
. DISP

Reg. Nº  TIPO_PROD                     COSTO_PROD ID_PROD NOM_PROD
     1   CLAVOS                              0.05       1 CLAVO1

. SKIP
. DISP     etc.
```

Los registros mostrados se agruparán por el mismo código de identificación de proveedor, después aparecerá el siguiente código de iden-

tificación tal y como aparece en el fichero de índice.

Repita este proceso introduciendo un código de identificación que no se use. Aparecerá un mensaje de NO ENCONTRADO. Los campos de índice de un solo campo de clave son los más fáciles de usar. Si se requieren campos múltiples, el comando INDEX debe incluir los campos en el orden en el que se va a formar el índice. Se puede crear un índice por tipo de producto, identificación del proveedor y nombre de producto, para la base de datos de productos, de la siguiente forma:

```
. USE PRODUCTO
. INDEX ON TIPO_PROD + ID_PROV + NOM_PROD TO PRODIDX2
. SET INDEX TO PRODIDX2          * SET INDEX TO se usa para cambiar el índice activo.
```

Introduzca ahora el comando FIND como sigue:

```
. FIND MARTILLO 3 MARTILLO1
```

Aparecerá el comando No Existe (NOFIND) porque el índice está estructurado de acuerdo a los tamaños de los campos. Intente ahora el comando FIND escribiéndolo así:

```
. FIND MARTILLO                 3 MARTILLO1
```

Ahora debe aparecer el punto; teclee DISP y aparecerá el registro actual. El uso de un índice compuesto puede acelerar la búsqueda de un registro determinado; sin embargo, requiere que la información usada por el FIND se complete con blancos hasta la longitud del campo. Otro problema del comando compuesto es que requiere que el primer campo en la lista de campos clave, esté siempre presente. Usando el índice compuesto PRODIDX2 será posible buscar por:

```
. FIND MARTILLO                                 Campo(1)
. FIND MARTILLO                 3               Campo (1+2)
. FIND MARTILLO                 3 MARTILLO1     Campo(1+2+3)
```

El comando de error NO EXISTE (NOFIND) aparecerá con los siguientes comandos:

```
. FIND 3                                        Campo(2)
. FIND MARTILLO1                                Campo(3)
. FIND 3 MARTILLO1                              Campo(2+3)
```

El fichero de índice es como una ventana en la base de datos que permite al operador una visión de datos seleccionados. Varias operaciones pueden alterar registros en la base de datos o el valor del campo clave usado para los ficheros de índice. Las operaciones involucradas son:

APPEND
EDIT
REPLACE
PACK (después de DELETE)

Use la base de datos NOMBRES con las siguientes entradas:

```
Reg. Nø NOMBRE          NOMBRE        EDAD
      1 JAIME           FERNANDEZ       29
      2 JORGE           HERNANDEZ       56
      3 ANDRES          RAMIREZ         53
      4 JORGE           FERNANDEZ       49
      5 RAMON           DIAZ            25
```

Indexe esta base de datos por NOMBRE, APELLIDO y EDAD, de esta forma:

```
. USE NOMBRES
. INDEX ON NOMBRE TO IDXNOM
. INDEX ON NOMBRE + APELLIDO TO IDXAPE
. INDEX ON EDAD TO IDXEDAD
```

Después de cada comando aparecerá el mensaje:

```
6 Registros indexado(s)
```

seguido del punto.

Edite ahora el registro 5 con el índice IDXNOM en uso:

```
. USE NOMBRES INDEX IDXNOM
. EDIT 5
```

En la pantalla aparecerá el registro 5 así:

```
NOMBRE      RAMON
APELLIDO    DIAZ
EDAD        25
```

El fichero de índice IDXNOM está relacionado con el campo NOMBRE que, si lo cambiamos a JOSE, quedará reflejado en el fichero de índice. Introduzca JOSE como nuevo valor de NOMBRE y pulse después Ctrl-W para escribir los cambios en el disco. Liste ahora los registros de la base de datos:

```
. LIST

Reg. Nø NOMBRE          NOMBRE        EDAD
      3 ANDRES          RAMIREZ         53
      1 JAIME           FERNANDEZ       29
      2 JORGE           HERNANDEZ       56
      4 JORGE           FERNANDEZ       49
      5 RAMON           DIAZ            25

. SET INDEX TO IDXAPE
. LIST

Reg. Nø NOMBRE          NOMBRE        EDAD
      3 ANDRES          RAMIREZ         53
      1 JAIME           FERNANDEZ       29
      4 JORGE           FERNANDEZ       49
      2 JORGE           HERNANDEZ       56
      5 RAMON           DIAZ            25
```

```
. USE NOMBRES INDEX IDXNOM
. SET INDEX TO IDXAPE, IDXEDAD
. SET INDEX TO IDXAPE
. LIST

Reg. Nº NOMBRE          NOMBRE          EDAD
      3 ANDRES          RAMIREZ           53
      1 JAIME           FERNANDEZ         29
      4 JORGE           FERNANDEZ         49
      2 JORGE           HERNANDEZ         56
      5 RAMON           DIAZ              25
```

La inclusión del comando SET INDEX TO permite actualizar los ficheros de índice incluidos en el comando, cuando se ejecute un comando EDIT. Si el campo de clave que se cambia durante el EDIT se usa solamente en un fichero de índice, el comando SET INDEX TO se puede omitir. Por ejemplo, si se va a cambiar el campo EDAD, como es índice único, solo necesita usar el índice IDXEDAD.

En modo APPEND, todos los ficheros índice deben ser actualizados. La siguiente secuencia de comandos permite añadir registros nuevos y actualizar los ficheros de índice.

```
. USE NOMBRES
. SET INDEX TO IDXNOM, IDXAPE, IDXEDAD
. APPEND
```

Añada el siguiente registro:

```
NOMBRE      ANGEL         Debe ser el segundo registro si lo listamos por nombre (IDXNOM)
APELLIDO    MARTINEZ      Debe ser el segundo registro si lo listamos por (IDXAPE)
EDAD        64            Debe ser el último registro si lo listamos por EDAD (IDXEDAD)
```

Use Ctrl-W para escribir los cambios a disco.

```
. LIST

Reg. Nº NOMBRE          NOMBRE          EDAD
      3 ANDRES          RAMIREZ           53
      6 ANGEL           MARTINEZ          64
      1 JAIME           FERNANDEZ         29
      2 JORGE           HERNANDEZ         56
      4 JORGE           FERNANDEZ         49
      5 RAMON           DIAZ              25
```

Seleccione ahora el índice IDXEDAD:

```
. SET INDEX TO IDXEDAD
. LIST

Reg. Nº NOMBRE          NOMBRE          EDAD
      5 RAMON           DIAZ              25
      1 JAIME           FERNANDEZ         29
      4 JORGE           FERNANDEZ         49
      3 ANDRES          RAMIREZ           53
      2 JORGE           HERNANDEZ         56
      6 ANGEL           MARTINEZ          64
```

El efecto de un comando REPLACE es el mismo si se usa el comando SET INDEX TO

```
. USE NOMBRES INDEX IDXNOM
. FIND ANDRES
. REPLACE NOMBRE WITH 'AAAA', EDAD CON 23
     1 Registro sustituido(s)
. LIST

Reg. Nº NOMBRE           NOMBRE         EDAD
     3  AAAA             RAMIREZ          23
     6  ANGEL            MARTINEZ         64
     1  JAIME            FERNANDEZ        29
     4  JORGE            FERNANDEZ        49
     2  JORGE            HERNANDEZ        56
     5  RAMON            DIAZ             25

. SET INDEX TO IDXEDAD
. LIST
```

Como IDXEDAD no estaba incluido en el comando SET INDEX TO, el listado no reflejará este cambio.

Finalmente, el comando PACK elimina los registros marcados para borrar, ya sea por el comando DELETE RECORD o usando Ctrl-U en modo EDIT.

```
. USE NOMBRES
. SET INDEX TO IDXAPE, IDXEDAD
. DELETE RECORD 6
. PACK
```

Aparecerán en la pantalla los siguientes mensajes:

```
     5 Registros copiado(s)
Reconstruyendo el índice A:IDXAPE.NDX
     5 Registros indexado(s)
Reconstruyendo el índice A:IDXEDAD.NDX
     5 Registros indexado(s)
```

Cuando se introducen datos dentro de una base de datos sin ficheros índice en uso, los ficheros de índice se pueden reconstruir usando el comando REINDEX:

```
. USE NOMBRES
. SET INDEX TO IDXNOM, IDXAPE, IDXEDAD
. APPEND
. REINDEX
```

En el paso APPEND añadimos uno o dos registros a la base de datos NOMBRES y continuamos después con la secuencia de comandos. Cuando se complete el comando REINDEX, liste el fichero por cada índice:

```
. USE NOMBRES INDEX IDXNOM
. LIST
. SET INDEX TO IDXAPE
. LIST
. SET INDEX TO IDXEDAD
```

Cada una de las nuevas entradas ha sido añadida al fichero índice y se lista en la posición correcta.

Los comandos usados para moverse por la base de datos operarán cuando esté en uso un fichero de índice; sin embargo, no reflejarán el orden del índice:

```
. USE NOMBRES INDEX IDXEDAD
. GO BOTTOM
. DISP

Reg. Nø NOMBRE              NOMBRE         EDAD
     2  JORGE               HERNANDEZ        56

. SKIP -2

Reg. N      6

. DISP

Reg. Nø NOMBRE              NOMBRE         EDAD
     6  ANTONIO             DIAZ             35
```

El efecto de este comando es el movimiento dentro de la base de datos, pero los registros mostrados no estarán por orden de número de registro.

La elección de un campo de clave para ficheros de índice compuestos requiere mucho cuidado. La función TRIM no se debe usar como parte de la clave del índice porque los datos usados no corresponderán después a ningún campo y pueden tener un tamaño aleatorio. Si se usan las funciones SUBSTR ($ en dBaseII) o STR (ver Capítulo 5) como parte de la clave, deben tener los números como literales. Los campos lógicos y los campos MEMO de dBaseIII no se pueden usar como claves de índice. El campo DATE del dBaseIII se puede usar como campo de índice.

```
. USE NOMBRES
. INDEX ON SUBSTR(NOMBRE,1,1) + APELLIDO TO IDXNOM1
```

o, en dBaseII

```
. INDEX ON $(NOMBRE,1,1) + APELLIDO TO IDXNOM1
```

Se creará el fichero de índice IDXNOM1 usando como clave el primer carácter del campo NOMBRE junto con el campo APELLIDO.

Si se usa un campo numérico como parte de un índice compuesto, se debe convertir a tipo carácter con la función STR:

```
. USE NOMBRES
. INDEX ON APELLIDO + STR(EDAD,3) TO IDXED1
. SET INDEX TO IDXED1
. LIST
```

Observe que el orden es correcto para APELLIDO.

Todos los ficheros de índice usados hasta ahora proporcionan listados en orden ascendente.

Puede resultar práctico establecer un índice que nos de un listado en orden descendente:

```
. USE NOMBRES
. INDEX ON - EDAD TO IDXEDM
. LIST
```

La lista que aparece ahora en la pantalla está en orden descendente por edad. Se puede hacer lo mismo con campos de caracteres, para alterar el orden en el que mostrará las entradas el comando LIST.

Además del comando FIND, el dBaseIII tiene un nuevo comando que se puede usar con bases de datos indexadas, es el comando SEEK. Este comando asume que su argumento es una variable, ej. una variable en memoria. Si se usa una cadena de caracteres como argumento, se debe encerrar entre comillas:

```
. USE PRODUCTO INDEX PRODIDX1
. STORE "CLAVOS" TO PROD1
. SEEK PROD1
```

o

```
. SEEK "CLAVOS"
```

En este tipo de acción, el dBaseII debe usar el comando FIND con CLAVOS encerrado entre comillas.

```
. FIND "CLAVOS"
```

Si el objetivo del comando SEEK es un valor numérico, se puede usar sin comillas y puede ser una variable numérica.

Si SEEK no encuentra un registro que coincida, aparecerá el mensaje NO EXISTE y se pondrá la función EOF como verdadera (.T.).

```
. USE PRODUCTO INDEX PRODIDX1
. SEEK "DESTORNILLADOR"
No Existe
.? EOF()
.T.
```

El dBaseIII tiene un campo nuevo, el campo DATE. Este campo permite introducir fechas de forma normal y convertirlas en fecha Juliana, que es numérica. Por desgracia el comando FIND no funciona con una base de datos indexada por un campo DATE. El comando SEEK permite usar fechas como argumentos, con alguna conversión. dBaseIII tiene algunas funciones nuevas que permiten la conversión de fecha en caracteres a fecha Juliana y viceversa. Para crear el fichero de índice requerido:

```
. INDEX ON FECHA TO IDXFECHA
```

para encontrar un registro por fecha se debe usar la siguiente secuencia:

```
. STORE "06/06/86" TO FECHA
. STORE CTOD(DATE) TO FECHA1
. SEEK FECHA1
```

La función CTOD convierte la fecha al formato Juliano requerido, y el comando SEEK la acepta por medio de la variable FECHA1. Se debe tener en cuenta que la parte numérica de la clave requiere aún el uso de la función STR cuando se usan elementos de tipo carácter y numérico para formar una clave compuesta.

LOCATE

Cuando la base de datos tiene un fichero índice relacionado con los datos a los que se quiere acceder, el comando FIND es más rápido. Si se requiere encontrar un registro de datos para el que no existe índice, se puede usar un comando LOCATE.

```
. USE NOMBRES
. LOCATE FOR APELLIDO = 'RAMIREZ'
Registro =        3
. DISP

Reg. Nº  NOMBRE             NOMBRE          EDAD
      3  AAAA               RAMIREZ           23

. LOCATE FOR APELLIDO = ' '          - entrada en blanco
Fin del ámbito (SCOPE) del LOCATE   - encontró el fin del fichero
```

Si LOCATE no encuentra un registro, genera en la pantalla el mensaje que hemos visto. Si se usa solamente entradas desde el teclado, la forma del comando que se ha visto es el adecuado. Cuando se está ejecutando un programa, es preferible que el valor del campo usado por el comando LOCATE se almacene en una variable en memoria.

```
. USE NOMBRES
. STORE 'RAMIREZ' TO APELL
. LOCATE FOR APELLIDO = APELL
Registro =        3
. DISP
```

Mediante el uso de operadores lógicos, el comando LOCATE nos proporciona acceso a los registros, de acuerdo a un criterio de selección:

```
. USE NOMBRES
. LOCATE FOR APELLIDO = 'RAMIREZ' .AND. EDAD = 23
Registro =        3
. DISP
```

Como ya indicamos anteriormente, se pueden usar variables para almacenar los valores seleccionados. Esto no es muy importante cuando usamos el teclado para la entrada de comandos. Sin embargo, bajo control de programa, las variables se usan de esta forma:

```
. USE NOMBRES
. STORE 'DIAZ' TO MNOMBRE
. STORE 20 TO MEDAD
. LOCATE FOR NOMBRE = MNOMBRE .AND. EDAD > MEDAD
Registro          5
. DISP
```

Observe que las variables en memoria que hemos usado en este ejemplo tienen el prefijo M, una práctica muy útil cuando se escriben programas largos. En el comando LOCATE se puede usar la función SUBSTR ($ en dBaseII) y cualquier función de selección.

Cuando el comando LOCATE encuentra un registro que coincide con la selección, puede que no sea el registro requerido, si existen múltiples entradas. Para continuar la búsqueda se debe poner a continuación el comando CONTINUE. dBase encontrará el siguiente registro que satisfaga el criterio de selección del último comando LOCATE:

```
. USE NOMBRES
. LOCATE FOR EDAD > 35
. DISP
. CONTINUE
```

Si el comando LOCATE o el CONTINUE no encuentran un registro que coincida, la función EOF() se pondrá a .T.. Es importante recordar que cuando el comando LOCATE encuentra un registro, el apuntador de registros comienza por esa posición en los comandos sucesivos. Si se usa un fichero muy grande, el comando GO TOP nos mandará al principio del fichero, permitiendo al comando LOCATE buscar en el fichero entero. Es posible restringir el rango de un comando LOCATE usando la opción NEXT dentro de la línea del comando:

```
. LOCATE NEXT 5 FOR TIPO_PROD = "CLAVOS"
```

que limitará la búsqueda a los 5 registros siguientes.

SORT

Los sistemas de índices y el comando LOCATE no alteran la estructura física de una base de datos. El registro 00001 permanece como primer registro, no importa cómo se usen los comandos INDEX y LOCATE. En algunas circunstancias puede ser deseable presentar una base de datos ordenada antes de su uso. dBaseII tiene un comando SORT:

```
. USE NOMBRES
. SORT ON APELLIDO TO NOMBRES1
```

El formato básico del comando en dBaseII es SORT ON campo TO fichero-nuevo. El SORT puede ser ascendente o descendente.

```
. USE NOMBRES
. SORT ON NOMBRE TO NOMBRES1 ASCENDING
```

En dBaseII se puede hacer SORT sucesivamente en una serie de campos, empezando con el campo de menor importancia y siguiendo hasta el de más importancia. Cada paso de la operación requiere su propia línea de comando.

El comando SORT en dBaseIII ha sido extendido significativamente para permitir operar co una lista de campos con el orden especificado en cada campo. En dBaseIII se han incorporado dos nuevas facilidades al comando SORT; estas son, la opción SCOPE y una cláusula FOR. Por ejemplo, para clasificar la base de datos NOMBRES por orden de edad y

alfabéticamente dentro de cada grupo de edades, se debe usar un comando como este:

```
. SORT TO NOMBRES1 ON EDAD/D,NOMBRE
```

La adición de una cláusula FOR puede restringir esta clasificación a aquellos registros para los que el campo EDAD satisfaga ciertas condiciones:

```
. SORT TO NOMBRES ON EDAD/D,NOMBRE FOR EDAD < 35
```

El comando SORT en dBaseIII tiene ahora un formato diferente del de dBaseII, ya que las opciones NEXT y FOR permiten restringir el uso a un rango de registros o a grupos de registros especificados por la opción FOR. El comando SORT es lento si se usa en ficheros grandes y si se tiene que clasificar el fichero completo. Requiere espacio en el disco con el que estamos trabajando o que el disco de destino tenga espacio libre como del tamaño del fichero que estamos clasificando. Esto es necesario porque la operación SORT crea una copia clasificada del fichero en uso.

Los comandos descritos hasta ahora permiten crear bases de datos, añadir datos y borrarlos, y encontrar datos. Existen varios comandos para cambiar los datos en todos o en algunos registros en la base de datos activa.

REPLACE

REPLACE es un comando que se usa para reemplazar datos en campos especificados:

```
. USE NOMBRES
. REPLACE APELLIDO WITH 'PARDO'
```

Este comando cambiará PARDO el valor que hay en el campo APELLIDO del primer registro. Se puede cambiar más de un campo con un comando REPLACE simple:

```
. USE NOMBRES
. STORE 'MARTINEZ' TO MFIND
. FIND MFIND
. REPLACE APELLIDO WITH 'PARDO', EDAD WITH 36
```

Si se cambia un registro de clave con el comando REPLACE, el fichero índice correspondiente, si está en uso, se ajustará de acuerdo con el nuevo valor. Nota: como se ha cambiado el orden del índice, el orden de registros, como se ve si se usa un comando LIST, reflejará este cambio.

```
. USE NOMBRES INDEX IDXNOM
. LIST
```

El comando REPLACE en dBaseII/III comprueba si hay claves alteradas y actualiza el fichero índice de acuerdo con el cambio. Esto puede disminuir la velocidad del proceso. Si se sabe que no se va a cambiar ningún registro de clave, es posible evitar el reindexar en dBaseII

mediante el uso de la opción NOUPDATE.

```
. USE NOMBRES INDEX IDXNOM
. REPLACE EDAD WITH 23 [NOUPDATE]
```

Un aspecto más del comando REPLACEm que debe ser mencionado. Si se usan bases de datos primarias y secundarias, el comando REPLACE solo se puede usar en la base de datos seleccionada. (Ver más adelante las opciones PRIMARY y SECONDARY. Por último, si un campo dado va a ser reemplazado a lo largo de la base de datos, se debe usar el comando REPLACE ALL. Por ejemplo, si la base de datos tiene un campo de PRECIO y el precio general aumenta un 10%, se puede hacer de la siguiente forma:

```
. REPLACE ALL PRICE + 1.1
```

Si se omite el ALL, se cambia el registro actual, esto se puede hacer así:

```
. REPLACE PRECIO WITH PRECIO*1.1 FOR ITEM='ABRIGO'
```

que reemplazará todos los precios de los registros que contengan A-BRIGO. Una alternativa de la expresión FOR es la expresión WHILE:

```
. REPLACE PRECIO WITH PRECIO*1.1 WHILE PRECIO<5700
```

que cambiará todos los precios menores de 5700 a 5700 + 10% de 5700, dando un incremento general del 10%.

CHANGE

El comando REPLACE es una forma eficiente de cambiar el valor de un campo entero; sin embargo, se suele dar el caso de tener que cambiar parte de un campo solamente. Supongamos que tenemos que reiniciar una serie de campos de fecha cambiando solamente la parte del año. El subcomando CHANGE nos permite hacerlo de la siguiente forma, en dBaseII:

```
. USE NOMBRES
. CHANGE FIELD FECHAN
RECORD: 00001
FECHAN 05/11/47
CHANGE? 47
TO 49
FECHAN 05/11/49
CHANGE? <RETURN>
```

De esta forma nos permitirá cambiar el campo FECHAN de cada registro. Podemos hacer que el comando sea más selectivo:

```
. USE NOMBRES
. CHANGE FIELD FECHAN FOR EDAD > 30
```

dBaseII buscará el primer registro que satisfaga la condición FOR y le dará la opción de cambiar el campo FECHAN como lo ha hecho antes. Se puede cambiar más de un campo con un solo comando CHANGE, por

ejemplo:

```
. CHANGE FIELD NOMBRE, FECHAN, FOR EDAD > 25
```

Con este comando buscará el primer registro donde el campo EDAD sea mayor de 25 y permitirá al operador cambiar los campos NOMBRE y FECHAN. Se puede borrar un campo completo tecleando Ctrl-Y y se puede salir del modo CHANGE pulsando la tecla Esc.

En dBaseIII cambia el formato en que presenta el campo cuando se elige el modo CHANGE, presentando el nombre del campo con todo el contenido del primer registro, permitiendo mover el cursor dentro para modificar su contenido. En este modo aparece también la cabecera de ayuda que ya vimos en el comando BROWSE, usando las mismas teclas para las diversas funciones permitidas.

```
. USE NOMBRES
. CHANGE FIELDS FECHAN
Reg Nº     1
FECHAN     05/11/47

Ctrl-W para salvar los cambios
```

Suele ser muy útil contar el número de entradas que satisfacen una determinada condición o el número total de piezas en almacén mediante la suma de un mismo campo a través de una base de datos. Para hacer informes puede ser útil reunir los registros de dos bases de datos para formar una tercera o actualizar la base de datos principal desde otra subsidiaria. El dBase tiene un grupo de comandos que permite hacer todo esto de una forma simple.

En dBaseIII se puede editar un campo MEMO con el comando CHANGE colocando el cursor en el campo MEMO y tecleando ^PgDn para entrar en el campo y modificarlo. La edición se realiza moviendo el cursor por el texto mediante las teclas de control, que son las mismas que las del comando MODIFY.

COUNT

El comando COUNT nos da el número de registros en la base de datos seleccionada. Si se le añaden cláusulas de cualificación, el número de registros que nos de, será el de los que satisfagan esas condiciones.

```
. USE NOMBRES
. COUNT                       * cuenta todos los registros
. COUNT FOR EDAD > 35         * cuenta los registros con EDAD > 35
. COUNT NEXT 3 FOR EDAD > 20
```

La última línea de comando nos dará el número de registros dentro de los 3 siguientes, que satisfacen la condición de que el campo EDAD sea mayor de 20.

Es posible almacenar en una variable de memoria el valor resultante.

```
. COUNT NEXT 6 FOR EDAD < 90 TO NUM1
. COUNT NEXT 6 WHILE EDAD < 90 TO NUM1
```

También es posible incluir una combinación de condiciones en la cláusula FOR, mediante operadores lógicos:

```
. COUNT NEXT 10 FOR EDAD < 90 .AND. NOMBRE="JORGE"
```

El formato general del comando es:

```
COUNT <ámbito> [FOR <expresión>]  [TO <variable>]
               [WHILE <expresión]
```

SUM

El comando SUM opera con datos numéricos del fichero en uso. El comando puede ser selectivo, mediante el uso de la opción FOR:

```
. USE PRODUCTO
. SUM COSTO_PROD FOR ID_PROD=1
. ? COSTO TOTAL DE LAS UNIDADES DEL PROVEEDOR 1
. SUM COSTO_PROD TO TOTAL FOR ID_PROD <> 1
. DISP MEMO
```

La segunda forma del comando permite escribir el valor en una variable en memoria, DISPLAY MEMO mostrará el valor de esa variable. Se puede listar más de un campo con el comando SUM. Como en el comando COUNT, la condición FOR se puede sustituir por WHILE. Se pueden listar hasta cinco campos y almacenarlos en variables en memoria mediante el comando TO. Si no se imponen condiciones, usarán todos los registros no borrados de la base de datos en uso. El comando SUM en dBaseII opera de la misma forma.

TOTAL

El comando TOTAL permite totalizar en otra base de datos los campos numéricos de los registros de la base de datos en uso. Por ejemplo, una base de datos de artículos que hemos recibido, contiene el número de entradas de un mes sobre la cantidad total recibida, identificados por un número de producto. Para resumir toda esta información, la base de datos de artículos recibidos se puede totalizar por número de artículo dando la cantidad total recibida. Asumiendo que le base de datos de origen tiene solamente dos campos, NUM_ARTIC y CANTIDAD:

```
. TOTAL ON NUM_ART TO SUMARIO FIELDS CANTIDAD     * en dBaseIII
. TOTAL ON NUM:ART TO SUMARIO CANTIDAD            * en dBaseII
```

Con este comando conseguiremos que se sumen todas las entradas individuales por número de artículo, y almacenar el resultado sobre un registro de la base de datos SUMARIO. La base de datos sobre la que se realiza el total requiere ser ordenada o indexada previamente por el campo usado para totalizar. El comando se puede modificar para que sea más selectivo, mediante la adición de las opciones FOR o WHILE.

```
. USE PRODUCTO INDEX PROIDX
. TOTAL ON ID_PROV TO SUMARIO FIELDS COSTO_ART FOR ID_PROV<>1
```

NOTA: PRODIDX es el fichero índice correspondiente al campo ID_PROV.

La base de datos de destino se crea si no existe; su estructura será la misma que la de la base de datos activa. Si existe, se escribirá sobre ella, usando dBaseIII. Si se usa dBaseII y la base de datos existe, su estructura permanecerá intacta y se usará para decidir qué campos serán totalizados aritméticamente, suponiendo que se omita la lista de campos. Un uso interesante del comando TOTAL es para eliminar campos duplicados. Si se incluye un campo no numérico en la lista de campos, se totalizarán sin generar mensaje de error, aunque no pasarán a la base de datos de totalización.

AVERAGE

AVERAGE es un nuevo comando de dBaseIII y calcula la media de los campos numéricos especificados de la base de datos activa.

```
. USE NOMBRES
. AVERAGE EDAD FOR NOMBRE="JORGE'
```

Este comando calculará la media de edad de todos los registros en los que el campo NOMBRE se igual a JORGE.

Los valores calculados por el comando AVERAGE se pueden almacenar en una variable de memoria:

```
. AVERAGE EDAD FOR NOMBRE="JORGE' TO EDAD1
. ? EDAD1

EDAD1     PUB  N     45
```

Se sacará la media de todos los campos numéricos a menos que se restrinja mediante una lista de campos.

UPDATE

El comando UPDATE se usa para modificar la base de datos seleccionada con información de otra base de datos. Consideremos una base de datos de ARTICULOS que contiene los campos NUM_ART, COSTO, ITEMS e ID_PROV. Esta base de datos se tiene que actualizar desde la base de datos de artículos recibidos (ART_REC) para mantener la base de datos ARTICULOS. La base de datos ART_REC contendrá los campos NUM_ART, COSTO y ITEMS. Tenga en cuenta que los nombres de los campos, sus tamaños y tipos deben coincidir con la estructura de la base de datos del fichero que vamos a actualizar.

En dBaseII, el comando UPDATE se usa así:

```
. SELECT SECONDARY
. USE ART:REC INDEX IDXARTR
```

```
. SELECT PRIMARY
. USE ARTICULOS INDEX IDXART
. UPDATE FROM ART:REC ON NUM:ART ADD ITEMS REPLACE COST
```

Los campos actualizados pueden ser sumados o reemplazados. Se actualiza un registro cuando coincide el NUM:ART de la base de datos de origen con el campo del mismo nombre de la base de datos seleccionada. En general, los campos que se van a usar se identifican con la palabra ON. Esta forma del comando UPDATE requiere que las bases de datos actual y la de origen de los datos, estén clasificadas por la clave usada para la comparación. La base de datos seleccionada puede estar indexada por la clave. El comando UPDATE comienza las dos bases de datos desde el principio. Si hay una coincidencia , se ejecutan las acciones ADD y REPLACE. Si no hay coincidencia, se usará el siguiente registro del fichero FROM. Este comando tarda algún tiempo en completarse, ya que cada registro del fichero FROM será comparado con el registro actual del fichero seleccionado mientras no ocurra una coincidencia, y los valores de la clave del fichero seleccionado sean menores que la del fichero FROM. Se puede usar una palabra de cualificación adicional [RANDOM] si la clave del fichero FROM es un solo campo que puede asumirse que coincidirá con una clave del índice del fichero seleccionado. La base de datos seleccionada debe estar indexada, pero los registros en el fichero FROM pueden estar en cualquier orden, ya que se lee cada uno de los registros y se ejecuta una búsqueda interna en la base de datos seleccionada.

```
. UPDATE FROM ART:REC ON NUM:ART RANDOM ADD ITEMS;
REPLACE COSTO
```

NOTA: 1) El punto y coma permite continuar un comando en la línea siguiente.
2) La opción REPLACE se puede modificar a REPLACE FIELD WITH campo. Ej. reemplazar un campo en la base de datos seleccionada con el campo correspondiente del fichero FROM.

Usando dBaseIII, el comando UPDATE tiene un formato ligeramente diferente. El comando usado para actualizar la base de datos ARTICULOS desde la base de datos ART_REC será:

```
. SELECT 2
. USE ART_REC INDEX IDXARTR
. SELECT 1
. USE ARTICULOS INDEX IDXART
. UPDATE ON NUM_ART FROM ART_REC REPLACE CANTIDAD WITH ;
CANT+ART_REC->CANTIDAD, COSTO WITH ART_REC->COSTO
```

En esta forma del comando, no se usa la opción ADD, se usa la sentencia REPLACE para realizar la suma. Para realizar el REPLACE WITH, el dBaseIII identifica la fuente de los datos con el nombre del fichero y el nombre del campo, de esta forma:

```
NOMB_FICH -> NOM_CAMPO

ART_REC -> COSTO
```

Esto identifica el campo desde el que vienen los nuevos datos de COSTO como el campo COSTO de la base de datos ART_REC.

JOIN

Este comando permite mezclar dos bases de datos para formar una tercera de acuerdo con un criterio de selección. Nos referiremos a las dos bases de datos sobre las que se va a ejecutar el comando JOIN, como primaria y secundaria. Más abajo se muestra el procedimiento usado para identificar las bases de datos primaria y secundaria. El comando SELECT PRIMARY se usa seguido por el JOIN. JOIN posiciona el apuntador de registro en el primer registro del fichero primario y, cada registro del fichero secundario se comprueba contra el criterio de selección. Si la comprobación devuelve un valor verdadero, ese registro se añade a la nueva base de datos.

```
. SELECT SECONDARY
. USE BASEDAT1
. SELECT SECONDARY
. USE BASEDAT2
. JOIN TO NUEVFICH FOR NUM:ART=S.NUM:ART FIELDS ;
lista de campos a añadir a la nueva base de datos
```

Cuando se hayan comprobado todos los registros de la base de datos secundaria contra el primer registro de la primaria, se usa el segundo registro de la base de datos primaria y se repite el proceso. Por consiguiente, si las bases de datos primaria y secundaria son grandes, el comando tardará bastante tiempo en completarse. Si las condiciones para crear un registro en el nuevo fichero son muy generales, puede crecer fácilmente. Tenga la precaución de preparar un condición FOR esencial.

Si se omite la opción FIELDS en el comando, se añadirán al fichero de salida todos los campos de la base de datos primaria. Se añaden tantos campos como tenga el fichero secundario, sin exceder de 32.

En dBaseIII el proceso es similar, excepto por la forma de nombrar las bases de datos de origen. El dBaseIII identifica las bases de datos activas por el número usado en el comando SELECT.

```
. SELECT 1
. USE BASEDAT1
. SELECT 2
. USE BASEDAT2
. SELECT 1
. JOIN WITH BASEDAT2 TO NUEV_FICH FOR (condiciones) ;
lista de campos a añadir a la nueva base de datos.
```

Los campos de la base de datos seleccionada se identifican por su nombre de campo solamente, Los campos de la base de datos del área secundaria se identifican por el nombre de fichero/nombre de campo según la convención que hemos visto en el anterior comando.

Control por Programa y Fichero de Comandos

Los comandos que hemos visto hasta ahora se han introducido directamente desde el teclado. Para realizar tareas útiles es más económico si estos comandos se pueden agrupar en ficheros de comandos. Los ficheros de comandos se preparan con uno de los siguientes medios:

1) con un editor o un proceso de texto
2) usando ZIP (dBASEII de 8 bit)
3) usando el editor propio de dBaseII/III

El método más conveniente para crear ficheros de comandos es un proceso de texto. Su manejo sofisticado de texto facilita la creación, haciéndolo especialmente indicado para escribir y editar ficheros de comandos. El editor disponible en dBaseII/III se invoca mediante el comando:

```
  . MODIFY COMMAND fichero
o . MODI COMM fichero
```

En este editor, los comandos se pueden teclear directamente dentro del fichero sin las restricciones que se encuentran al usar ZIP.

Los principales comandos del editor son:

Comando	Función
^N	Inserta una línea en blanco en la posición del cursor
^T	Borra la línea sobre la que está el cursor y mueve las líneas inferiores hacia arriba.
^W	Escribe los cambios al fichero en disco y abandona el editor de MODIFY COMMAND.
^Q	Abandona el MODIFY COMMAND sin salvar los cambios realizados al fichero.
^R	Mueve el texto una línea hacia abajo
^C	Mueve una página de texto hacia arriba.

Hay algunas restricciones que se deben recordar cuando se usa este editor:

1) Las líneas deben de ser de 77 o menos caracteres
2) Los caracteres TAB pasan a ser espacios sencillos
3) El cursor solo puede retroceder 4000 caracteres
4) no tiene operaciones de búsqueda o bloque

En dBaseIII, el mismo comando permite al usuario entrar en el editor de texto. Este editor, en dBaseIII, tiene un rango mucho más amplio de funciones que el de dBaseII.

Comando	Función
Flechas	Mueven el cursor una posición en la dirección de la flecha.
Atrás	Borra el carácter a la izquierda del cursor
^A, HOME	Mueve el cursor al principio de la palabra

^C,PgDn	Mueve la pantalla 18 líneas hacia abajo.
^F,End	Mueve el cursor al principio de la siguiente palabra.
^KG,Del	Borra el carácter sobre el que está el cursor
^KR	Lee un fichero externo dentro del fichero editado
^KW	Escribe el fichero editado sobre otro fichero
^M,RETURN	Mueve el cursor al principio de la siguiente línea de texto o comienza una nueva línea.
^N	Inserta una línea en blanco después del cursor
^Q,Esc	Sale sin salvar los cambios
^R,PgUp	Mueve la pantalla 18 líneas hacia arriba
^T	Borra todos los caracteres desde la posición del cursor hasta el principio de la palabra siguiente.
^V,Ins	Cambia el modo inserción.
^W	Sale del editor y salva los cambios
^Y	Borra la línea sobre la que está el cursor.

Cuando se usa el MODIFY COMMAND, el dBaseIII intentará encontrar el fichero que se le suministra en el comando. Si no existe el fichero, lo crea nuevo. Si se edita un fichero existente, se almacena en disco una copia de seguridad. El mayor fichero que se puede editar es de 4096 caracteres. Por último, para producir una copia en impresora del fichero editado, use el comando:

```
. TYPE fichero TO PRINT
```

El editor de dBaseIII tiene la mayoría de las funciones que proporciona un proceso de texto y, aquellos que hayan usado previamente procesos de texto para el desarrollo de programas, no lo necesitarán para el dBaseIII.

Para ilustrar el uso de los ficheros de comandos y el control de programa, vamos a considerar el problema de mantenimiento de productos en una base de datos. El mantenimiento de una base de datos de este tipo precisará de:

1) Introducir nuevos datos
2) Modificar los registros existentes
3) Hacer informes de la base de datos.

Para introducir un nuevo registro en la base de datos, la secuencia más simple de comandos es:

```
. USE PRODUCTO
. APPEND BLANK
```

Observe el cambio del comando APPEND en esta situación.

Este fichero de comandos es el más simple posible. Abre el fichero PRODUCTO, añade un registro en blanco y permite al operador introducir un nuevo registro. Si embargo, hay dos problemas principales en ello: primero, solo permite añadir un registro, y segundo, la entrada de datos requiere que el operador tenga algún conocimiento de la estructura de los campos.

Para permitir que se repita el proceso, se requiere una situación de control. En dBase hay dos estructuras que lo realizan:

```
IF THEN ELSE ENDIF
```

o

```
DO WHILE ENDDO
```

La estructura de control de DO WHILE ENDDO es:

```
DO WHILE <condición>
   comandos
ENDDO
```

Considerando el problema de la entrada de datos en el fichero de PRODUCTO. La estructura DO WHILE ENDDO permitirá una entrada continua de la siguiente forma:

```
USE PRODUCTO
STORE 'S' TO CONT
DO WHILE CONT = 'S'
APPEND BLANK
CLEAR                          * ERASE en dBaseII
@10,10 SAY '¿Más datos?'
@10,30 GET CONT
READ
ENDDO
RETURN
```

Observe que este fichero de comandos hace varias cosas. Selecciona la base de datos correcta, inicializa la variable de control CONT y ejecuta el bucle de control DO WHILE ENDDO en el que el valor de la variable de control puede ser cambiado. El fichero de comandos permite añadir registros hasta que la variable CONT se cambie a N. El valor de CONT se comprueba en la sentencia ENDDO cada vez que se ejecuta el bucle de control. El fichero de comandos que contiene estos comandos se salva en un fichero llamado RECORDIN.CMD y se ejecuta mediante el comando:

```
. DO RECORDIN
```

El comando DO es con el que ejecuta ficheros de comandos el dBase Si se ejecuta el fichero de comandos que acabamos de crear, el efecto de alguno de los comandos aparecerá por la pantalla. Para evitarlo se usa el comando SET TALK OFF:

```
SET TALK OFF
CLEAR                     * en dBaseII reemplace CLEAR con ERASE
```

```
DO RECORDIN
```

SET TALK OFF inactiva la salida a la pantalla de ciertos comandos de dBase.

CLEAR limpia toda la pantalla y coloca el cursor en la parte superior izquierda.

ERASE es el equivalente a CLEAR en dBaseII, y tiene el mismo efecto.

Se suelen incluir los comandos SET TALK OFF y CLEAR o ERASE, al principio de los ficheros de comandos.

Si lo que se quiere es editar un registro, el fichero de comandos debe encontrar el registro correcto y editarlo.

Este fichero es un fichero básico de comandos que busca y edita un registro.

```
SET TALK OFF
CLEAR                          * Reemplácelo con ERASE para dBaseII
STORE '     ' TO MPROD         * se usa para buscar un registro
USE PRODUCTO INDEX PRODIDX
@10,10 'INTRODUZCA EL NOMBRE DEL PRODUCTO'
@10,44 GET MPROD
READ
FIND MPROD
EDIT
```

También se puede usar para ejecutar o no un grupo de comandos dependiendo del valor de una variable, CONT en este ejemplo.

```
IF CONT = 'S'
Comandos
ELSE                    * CONT <> 'S'
Comandos
ENDIF
```

Aquí no se usa la opción ELSE porque, a menos que CONT sea igual a 'S', no se ejecutará el comando STORE.

```
IF CONT = 'S'
STORE ' ' TO MPROD
ENDIF
```

Para permitir editar continuamente la base de datos de productos, se debe añadir un bucle de control y rellenar la variable MPROD con espacios. El bucle de comandos se prepara usando DO WHILE ENDDO con la variable de control CONT. La estructura IF ENDIF restaura la variable MPROD dependiendo del valor de de CONT.

```
SET TALK OFF
CLEAR                   * reemplazar por ERASE en dBaseII
STORE '     ' TO MPROD
USE PRODUCTO INDEX PRODIDX1
DO WHILE CONT = 'S'
    @10,10 SAY 'Introduzca el nombre del producto'
    @10,41 GET MPROD
```

```
    READ
    FIND MPROD
    EDIT
    CLEAR
    @10,10 SAY '¿Continuo? (S/N)
    @10,27 GET CONT
    READ
      IF CONT = 'S'
        STORE '     ' TO MPROD
      ENDIF
ENDDO
RETURN
```

El fichero de comandos CAMBIO.CMP ha crecido para permitir la edición continua. La variable CONT se usa para controlar la ejecución del bucle. La sentencia IF se usa para restaurar MPROD con blancos. Esta forma de la sentencia IF, se puede incluir la cláusula ELSE para liberar MPROD si no se va a necesitar en algún punto posterior del programa:

```
IF CONT = 'S'
STORE '     ' TO MPROD
ELSE
RELEASE MPROD            * MPROD desaparece de memoria
ENDIF
```

Creación de informes en dBaseII

El último requerimiento es un programa que obtenga informes. dBase-II tiene un comando REPORT para generar informes simples de una base de datos.

La capacidad de crear informes del dBaseII como una parte de su lenguaje de comandos, permite al operador generar informes de un base de datos basándose en el diálogo que tiene lugar después de lanzar el comando REPORT. La secuencia de comandos será como sigue:

```
USE PRODUCTO
REPORT
```

La pantalla mostrará una serie de preguntas.

```
ENTER REPORT FORM NAME: SUMARIO
ENTER OPTIONS M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH
PAGE HEADING(Y/N) Y
ENTER PAGE HEADING: SUMARIO DE PRODUCTOS
DOUBLE SPACE REPORT(Y/N) N
ARE TOTALS REQUIRED(Y/N) Y
SUBTOTALS IN REPORT(Y/N) N
COL      WIDTH: CONTENIDO
```

Se introducen el número de columna, su anchura y su contenido.

```
011   30   tipo:prod        tamaño y nombre del campo de la base de datos que se usará para
                            rellenar la primera columna.
```

```
ENTER HEADING: TIPO PRODUCTO La cabecera elegida debe ser de menor o igual longitud que el tamaño del
                             campo, ej. si el tamaño del campo es 30 caracteres, limite la cabecera a
                             30 caracteres.
```

Se introduce la definición de columnas hasta que no se necesiten más, terminando el diálogo con RETURN. En este momento se comienza a generar el informe y la impresora lo imprime. Si sale una página en blanco por la impresora antes de que comience a salir el informe, se puede eliminar insertando el comando SET EJECT OFF al principio de la secuencia que prepara el informe. La forma general del comando REPORT es:

```
REPORT FORM nombre_formulario [ámbito] [FOR expresión] [TO PRINT]
```

El comando REPORT asume como ámbito ALL, a menos que se especifique otro distinto.

El efecto de esta secuencia es generar un fichero que pueda ser llamado y usado.

Creación de informes en dBaseIII

En dBaseII, los informes se generan por medio de una secuencia de preguntas; el formato del informe se almacena en un fichero .FRM que se puede volver a usar cuando se necesite. En dBaseIII se generan los informes por medio de una serie de pantallas. El generador de informes se arranca por medio del comando:

```
CREATE REPORT SUMARIO1
```

Después de lanzar el comando, aparece una pantalla con el formato que vemos a continuación.

Asumiendo que el fichero seleccionado es PRODUCTO.DBF:

```
Estructura del fichero A:PRODUCTO.DBF
_____________________________________________________________________________
TIPO_PROD  C  30   |                  |                  |                  |
COSTO_PROD N   6  2|                  |                  |                  |
ID_PROD    N   3   |                  |                  |                  |
NOM_PROD   C  30   |                  |                  |                  |
___________________|__________________|__________________|__________________|

                          Cabecera de página:


                    Informe del sumario de productos

                           Hierros Román S.A.


                  Anchura pág. (nø de carac.):   80
                  Margen izqdo. (nø de carac.):   8
                  Margen dcho. (nø de carac.):    0
                  Nø de líneas/página:           58
                  ¿Informe a dos espacios? (S/N)  N
```

Este formato permite al usuario introducir la cabecera del informe y seleccionar las condiciones de impresión. Se usa la tecla RETURN para moverse de un campo al siguiente.

Cuando se han completado todas las entradas de este formato, se pulsa RETURN o PgDn, para situarse en la siguiente pantalla en secuencia, que pide los nombres de los campos de los que se requieren totales y subtotales. Si se van a obtener subtotales en un informe, la base de datos debe estar indexada o clasificada por el campo especificado como campo de subtotales. Después de la pantalla de control de cabecera y de opciones de impresión, aparece la pantalla de subtotales:

```
Estructura del fichero A:producto.dbf
________________________________________________________________________
TIPO_PROD  C  30  |                 |                 |
COSTO_PROD N   6 2|                 |                 |
ID_PROD    N   3  |                 |                 |
NOM_PROD   C  30  |                 |                 |
________________________________________________________________________
Grupo/subtotal según          COSTO_PROD



¿Resumen únicamente? (S/N)  N       ¿Cambio pág. tras grupo/subtotal? (S/N) N

Cabecera de Grupo/subtotal:   Costo de productos


Subgrupo/subsubtotal según:




Cabecera Subgrupo/subsubtotal:


-----------------------------------------------------------------------
```

Para moverse de un área de entrada de datos a otra, se usa la tecla RETURN, y para pasar a la siguiente pantalla PgDn. La siguiente pantalla se usa para definir los campos que se requieren en el informe:

```
Estructura del fichero A:PRODUCTO.DBF
________________________________________________________________________
TIPO_PROD  C  30  |                 |                 |
COSTO_PROD N   6 2|                 |                 |
ID_PROD    N   3  |                 |                 |
NOM_PROD   C  30  |                 |                 |
________________________________________________________________________
                                   Campo 1            Cols. libres =   72
>>>>>>>>---------------------------------------------------------------
```

```
Campo        COSTO_PROD
  contenido



           1 COSTO
Campo      2PRODUCTO
  cabecera 3
           4
Anchura      6                      Nº de decimales:   2 ¿Total? (S/N) N

----------------------------------------------------------------------------
```

Después de definir este campo (contenido del campo = nombre del campo de la base de datos activa) pulse RETURN y entre en la cabecera de campo. La cabecera de campo puede ser mayor que el ancho de la columna, pero el generador de informes la ajustará dependiendo del espacio libre. Para definir el resto de los campos requeridos para el informe, aparecerán sucesivas páginas (pulsando RETURN o PgDn después de haber definido la cabecera) como la anterior. El único cambio es la línea que hay después de la descripción del formato de la base de datos, en la que aparecerán las cabeceras de los campos ya definidos en la posición que van a ocupar; como vemos a continuación:

```
Estructura del fichero A:PRODUCTO.DBF
____________________________________________________________________________
TIPO_PROD  C  30  |                 |                  |
COSTO_PROD N   6 2|                 |                  |
ID_PROD    N   3  |                 |                  |
NOM_PROD   C  30  |                 |                  |
____________________________________________________________________________
                                     Campo 2          Cols. libres =   63
>>>>>>>> COSTO    ----------------------------------------------------------
         PRODUCTO


           999.99

Campo        ID_PROD
  contenido



           1IDENTIFIC
Campo      2PROVEEDOR
  cabecera 3
           4
Anchura      9                      Nº de decimales:   0 ¿Total? (S/N) N

----------------------------------------------------------------------------
```

Para salvar el formato completo teclee ^End o pulse RETURN al sin introducir datos en el formato.

NOTA: Se pueden usar los comandos de BROWSE (flechas) para moverse

por los campos de la parte superior de la página del generador de informes.

Si se requieren totales en más de un campo, hace falta indexar la base de datos en el orden de los subgrupos de totales. Usando la base de datos PRODUCTO como ejemplo, para generar un informe con subtotales en COSTO_PROD y TIPO_PROD, hará falta crear un índice en ese orden. Recuerde que COSTO_PROD es un campo numérico y se debe convertir a cadena para crear un índice compuesto. Estas consideraciones nos demuestran que debemos tener cuidado en el diseño de las estructuras de la base de datos y del índice, si se requiere un informe complejo.

Después de crear el formato de un informe con el comando REPORT, se puede modificar para reflejar cambios posteriores mediante el comando MODIFY REPORT. Usando este comando se puede editar el informe original usando PgDn y PgUp para pasar de una página a otra. Se introducen los cambios usando las facilidades del editor de pantalla. Mediante el comando ^HOME se puede tener acceso más directo a partes del informe y obtener información adicional en una pantalla como esta:

```
Estructura del fichero A:PRODUCTO.DBF

TIPO_PROD  C  30  |          |          |
COSTO_PROD N   6 2|          |          |
ID_PROD    N   3  |          |          |
NOM_PROD   C  30  |          |          |

Titulo  Reunir  Primer  Central  Ultimo  Añadir  Guardar  Salir




Pasar a la pantalla de formateo de cabecera y página
```

La entrada 'Titulo' de la lista está resaltada y proporciona un breve explicación en la parte inferior de la pantalla . Cada opción se puede seleccionar por turno posicionando el cursor sobre ella, y aparecerá una breve explicación de su función en la parte inferior de la pantalla. Para seleccionar una opción pulse la tecla RETURN. Al final de la sesión de modificación, se pueden salvar los cambios usando ^END o pulsando la tecla Esc.

Para obtener un informe, el comando es el mismo que en dBaseII:

```
REPORT FORM (nombre formulario)
```

Si se requieren totales múltiples, se debe usar un fichero índice con una clave con los campos que se van a usar, y en el orden requerido.

Este fichero de comandos forma la base de un sistema manejado por medio de menús, que permite al operador seleccionar y ejecutar una opción. El menú será así:

```
     BASE DE DATOS DE PRODUCTOS

1>   INTRODUCIR UN PRODUCTO NUEVO
2>   EDITAR UN REGISTRO DE PRODUCTO
3>   INFORME
4>   SALIR
```

Este tipo de pantalla se crea mejor por medio del ZIP (vea el capítulo 6). Asumiremos que ya se ha preparado el fichero de comandos MENU.CMD. El menú realizará tareas de:

1) entrada de datos
2) cambios en los datos
3) informes

de forma resumida. Para que funcione este menú, deberemos tener una forma de seleccionar el fichero de comandos que se debe ejecutar. El siguiente fichero de comandos lo hace:

```
SET TALK OFF
ERASE
STORE 'S' TO CONT
STORE 'S' TO CONT1
STORE 0 TO ACCION
DO WHILE CONT='S'
     DO MENU1 - Muestra el menú en la pantalla
     @20,10 SAY 'Introduzca la acción requerida'
     READ
     DISPLAY selecciona pregunta y lee respuesta
     DO WHILE CONT1 = 'S'
          IF ACCION = 1
          DO RECORDIN
          ELSE
               IF ACCION = 2
               DO CAMBIO
               ELSE
                    IF ACCION = 3
                    REPORT FORM INFORME1 TO PRINT
                         ELSE
                         IF ACCION = 4
                         QUIT
                         ENDIF
                    ENDIF
               ENDIF
          ENDIF
   ENDDO
ENDDO
```

Las estructuras IF ELSE encadenadas son una forma de obtener la selección del fichero de comandos apropiado. La estructura DO WHILE ENDDO permite mostrar el menú después de la ejecución del fichero de comandos seleccionado. Observe que cada uno de los ficheros de comandos terminan con un comando RETURN. Como los ficheros de comandos son llamados desde el programa de menú, el control debe retornar a este

fichero por medio del comando RETURN. El comando RETURN hace que vuelva el control al programa que lo ha llamado. Una alternativa para la estructura IF ELSE ENDIF es la estructura CASE. Esta estructura reemplaza a la IF de la siguiente forma:

```
DO CASE
    CASE ACCION = 1
        DO RECORDIN
    CASE ACCION = 2
        DO CAMBIO
    CASE ACCION = 3
        REPORT FORM INFORME1 TO PRINT
    CASE ACCION = 4
        QUIT
```

Esta es la forma más simple de la sentencia CASE. A esta sentencia se le puede añadir la cláusula OTHERWISE. Por ejemplo, si se usa una entrada inválida en ACCION, se puede usar la cláusula OTHERWISE para permitir la corrección.

```
DO CASE
    CASE ACCION = 1
        DO RECORDIN
    CASE ACCION = 2
        DO CAMBIO
    CASE ACCION = 3
        REPORT FORM INFORME1 TO PRINT
    CASE ACCION = 4
        QUIT
OTHERWISE
    CLEAR
    @10,10 SAY 'Entrada inválida - Repita'
    @10,30 GET ACCION
    READ
ENDCASE
```

La utilidad de esta cláusula OTHERWISE se demuestra cuando aparece una situación de error. Otra forma de hacerlo es usando una sentencia CASE especial para procesar los casos especiales y la opción OTHERWISE para procesar la rutina. Si no se incluye la sentencia OTHERWISE o ninguna de las opciones son válidas, no sucede nada cuando se encuentra la sentencia CASE. No se deben insertar sentencias entre DO CASE y la primera opción CASE, porque no se ejecutará.

Los comandos que hemos visto en este capítulo están relacionados con la organización de las bases de datos para facilitar el acceso, búsqueda de datos, cambio de datos que ya están en la base de datos, obtener información de todos los registros de la base de datos, obtener informes del contenido de la base de datos e introducir elementos de control de programa usando las estructuras IF ELSE ENDIF, DO WHILE ENDDO y DO CASE ENDCASE.

CAPITULO 3

Operadores de dBaseII/III

Los comandos descritos en los capítulos anteriores se pueden usar para añadir, mostrar y cambiar datos dentro de una base de datos. Para producir programas útiles, se requiere un rango de comandos mucho más grande. Someramente se pueden clasificar estas funciones en:

- operadores lógicos.
- operadores aritméticos.
- operadores relacionales.
- operadores de cadena.
- funciones.

La primera sección de este capítulo tratará de los operadores y las funciones usadas en dBaseII y después se describirán los que han cambiado, los que se han añadido y los que se han eliminado en dBaseIII.

Operadores Lógicos

Los operadores lógicos disponibles en dBaseII son:

()	paréntesis para agrupar
.NOT.	NO lógico
.AND.	Y lógico
.OR.	O lógico
$	operador lógico de subcadena

Para ilustrar el uso de estos operadores, usaremos un ejemplo simple de una base de datos de productos:

PRODUCTO	COSTO	NOMBRE	IDENTIFICADOR PROVEEDOR
CLAVO	0.40	NORMAL	1
CLAVO	0.10	NORMAL1	2
CLAVO	0.25	NORMAL2	3
MARTILLO	875.00	MARTILLO1	1
MARTILLO	980.00	MARTILLO2	3
MARTILLO	1150.00	MARTILLO3	2
SIERRA	520.00	SIERRA1	1
SIERRA	710.00	SIERRA2	1
SIERRA	920.00	SIERRA3	2
TORNILLO	0.25	TORNILLO1	3
TORNILLO	0.30	TORNILLO2	3
TORNILLO	0.35	TORNILLO3	1

Si quiere ver todas las entradas de clavos y tornillos, necesita un comando que seleccione todos los registros para los que el campo PRODUCTO sea CLAVO o TORNILLO. Observe que del planteamiento de la cuestión se desprende el operador que hemos de usar, .OR.

Con los registros que hemos introducido en el fichero PRODUCTO, introduzca los siguientes comandos:

```
. USE PRODUCTO
. DISPLAY FOR TIPO_PROD = 'CLAVO' .OR. TIPO_PROD = 'TORNILLO'
```

En la pantalla aparecerán los siguientes registros:

```
TIPO_PROD  COSTO_PROD       NOM_PROD       ID_PROV

CLAVO         0.40          NORMAL         1
CLAVO         0.10          NORMAL1        2
CLAVO         0.25          NORMAL2        3
TORNILLO      0.25          TORNILLO1      3
TORNILLO      0.30          TORNILLO2      3
TORNILLO      0.35          TORNILLO3      1
```

De forma similar, el operador .AND. puede encontrar registros que tengan elementos comunes. Supongamos que desea buscar los registros de clavos y tornillos suministrados por el proveedor 1. Introduzca la siguiente línea:

```
. DISPLAY FOR (TIPO_PROD = 'CLAVO' .OR. TIPO_PROD = 'TORNILLO');
.AND. ID_PROV = 1
```

```
TIPO_PROD  COSTO_PROD       NOM_PROD       ID_PROV

CLAVO         0.40          NORMAL         1
TORNILLO      0.35          TORNILLO3      1
```

Los paréntesis son esenciales aquí para asegurarse de la operación correcta del comando. Intente introducir la siguiente línea:

```
. DISPLAY FOR TIPO_PROD = 'CLAVO' .OR. TIPO_PROD = 'TORNILLO'.AND.;
ID_PROV = 1
```

La pantalla resultante será bastante diferente de la anterior. Aparecerán todas las entradas de clavos y una sola entrada de tornillos. La razón para que suceda esto está en el orden de precedencia de los operadores lógicos. AND tiene más prioridad que OR, por lo que el comando se leería así:

```
. DISPLAY FOR TIPO_PROD = 'CLAVO' .OR. (TIPO_PROD = 'TORNILLO'.AND.;
ID_PROV = 1)
```

El operando .OR. divide el comando en dos sentencias con TIPO_PROD, en lugar de operar con los campos TIPO_PROD. Esto ilustra la importancia del uso de los paréntesis separando en partes las expresiones lógicas.

Si se necesita encontrar todas las entradas que no satisfagan un grupo determinado de condiciones, se puede usar el operador lógico .NOT.:

```
. DISPLAY ALL FOR .NOT. (TIPO_PROD = 'CLAVOS' .OR. TIPO_PROD = 'TORNILLOS')
```

Esta línea de comando nos dará una pantalla con todos los registros en los que TIPO_PROD no sea clavo ni tornillo:

TIPO_PROD	COSTO_PROD	NOM_P70D	ID_PROV
MARTILLO	875.00	MARTILLO1	1
MARTILLO	980.00	MARTILLO2	3
MARTILLO	1150.00	MARTILLO3	2
SIERRA	520.00	SIERRA1	1
SIERRA	710.00	SIERRA2	1
SIERRA	920.00	SIERRA3	2

Para conseguir este resultado hemos encerrado la condición .OR. entre paréntesis, de forma que .NOT. se aplique a todos los datos.

Otro ejemplo del uso del .NOT. lógico es:

```
. DISPLAY FOR (TIPO_PROD = 'CLAVOS' .OR. TIPO_PROD = 'TORNILLOS');
.AND. .NOT. ID_PROV = 1
```

Este comando nos mostrará los registros siguientes:

TIPO_PROD	COSTO_PROD	NOM_PROD	ID_PROV
CLAVO	0.10	NORMAL1	2
CLAVO	0.25	NORMAL2	3
TORNILLO	0.25	TORNILLO1	3
TORNILLO	0.30	TORNILLO2	3

Algunas veces resulta útil poder hacer una búsqueda de una cadena de caracteres dentro de un campo. Por ejemplo:

```
. DISPLAY FOR TIPO_PROD = $ 'MARTILLO        , SIERRA'
```

producirá el mismo resultado que:

```
. DISPLAY FOR TIPO_PROD = 'MARTILLO' .OR. TIPO_PROD = 'SIERRA'
```

La primera sentencia nos dará como resultado una pantalla con todos los registros en los que el campo TIPO_PROD esté contenido dentro de la cadena dada ($ 'MARTILLO ,SIERRA'). Los espacios adicionales se incluyen debido a que el ordenador mirará todo el campo exactamente. El efecto de un .NOT. lógico será la exclusión de todos los registros en los que el campo del producto contenga MARTILLO o SIERRA. El $ es una abreviatura de decir "contenido en" y se refiere normalmente a una operación de cadena. El uso de operadores lógicos permite la selección de registros en orden ascendente de acuerdo a un criterio seleccionado. Los operadores lógicos son particularmente útiles en la comprobación de la entrada de datos para eliminar errores.

Como ejemplo podemos poner la captura de errores en un menú de selección, usando:

```
1) añadir un registro
2) editar un registro
3) buscar un registro

Elija una operación.
```

Cuando aparece este menú en la pantalla, se introduce la opción requerida junto al mensaje 'Elija un operación'. Se puede comprobar la respuesta de esta forma:

```
STORE "C" TO VERIFICA
DO WHILE VERIFICA = "S"
DO MENU                             * ir a la pantalla del menú
IF .NOT. (ACCION = 1 .OR. ACCION = 2 .OR. ACCION = 3)
STORE "F" TO VERIFICA               * VERIFICA es un control
                                    * que permite volver a entrar
STORE " " TO ACCION
ENDIF
ENDDO
```

Otro ejemplo es la comprobación de fechas. La fecha se introduce como DD/MM/AA.

Los dos primeros caracteres deben estar entre 1 y 31, o 30, o 28, dependiendo del mes. Se debe hacer una comprobación especial para los años bisiestos. Los dos caracteres siguientes deben estar entre 1 y 12 y el tercer grupo debe representar el año.

Variables

Hasta ahora, todos los comandos de almacenamiento de datos han afectado a registros de bases de datos. Los datos contenidos en registros de una base de datos se almacenan permanentemente. Es útil disponer de variables transitorias en una aplicación. Se pueden considerar como un grupo de casilleros cuyo número puede variar, pero que tienen un máximo de 256 (64 en dBaseII). El valor y tipo de datos almacenados puede cambiar muchas veces durante la ejecución de un programa. Algunas de las variables pueden permanecer activas a lo largo de todo el programa, algunas puede que sean requeridas solamente durante parte del programa, y se borren después. Para crear una variable en memoria se usa el comando:

```
STORE 'A' TO ITEM1
A
```

donde A es el valor que se da a la variable ITEM1.

Teclee la siguiente secuencia de comandos:

```
. STORE 'ABC' TO ITEM1
ABC
. STORE  123  TO ITEM2
123
. STORE 'XYZ' TO ITEM3
XYZ

. ? ITEM1
ABC
                          El valor de la variable se imprime en
                          la siguiente línea. Observe que el valor
. ? ITEM2                 numérico se imprime precedido por espacios
     123
```

```
. ? ITEM3
XYZ

. DISPLAY MEMORY

ITEM1   pub   C   "ABC"
ITEM2   pub   N          123  (        123.00000000)
ITEM3   pub   C   "XYZ"

    3 variables definidas.        19 bytes usados
  253 variables disponibles.    5981 bytes disponibles

. RELEASE ITEM1

. DISP MEMO                      * Use la forma abreviada del
                                   comando DISPLAY MEMORY

ITEM2   pub   N          123  (        123.00000000)
ITEM3   pub   C   "XYZ"

    2 variables definidas.        14 bytes usados
  254 variables disponibles.    5986 bytes disponibles
```

Cuando se teclean los comandos STORE y se pulsa RETURN al final, el valor almacenado en la variable aparecerá en la pantalla. La sentencia PRINT debe mostrar cada variable en la pantalla, y el comando DISPLAY MEMORY hará que se imprima la lista completa de variables en memoria. Observe que la segunda vez que listamos las variables en memoria usamos el comando DISP MEMO, con las cuatro primeras letras de cada palabra del comando. La segunda lista de memoria contiene solamente dos variables, ya que el comando RELEASE borra el ITEM1 de la memoria.

La variables en memoria son transitorias, de tal modo que si el ordenador se apaga, se perderán todas las variables almacenadas en la memoria. Es posible salvar el contenido de la memoria para poder volver a usarlo con el comando:

```
. SAVE TO MEMFICH
```

Este comando salvará las variables en memoria a un fichero llamado MEMFICH, con la extensión .MEM. La entrada del directorio será MEMFICH.MEM. Se puede seleccionar el rango de variables salvadas mediante la adición de las condiciones adecuadas:

```
. SAVE TO MEMFICH ALL LIKE ABC****
```

Los asteriscos sustituyen a cualquier carácter que vaya a continuación.

o

```
. SAVE TO MEMFILE ALL LIKE A??DEF
```

Las interrogaciones enmascaran caracteres dentro de un nombre de variable y pueden ser cualquier carácter permitido.

La primera forma del comando se usa para salvar variables que comiencen con ABC y que tengan otros cuatro caracteres. La segunda forma salvará las variables que comiencen con A, terminen con DEF y tengan dos caracteres cualquiera entre la A y DEF.

El comando RELEASE se puede hacer también selectivo añadiéndole las condiciones adecuadas.

```
. RELEASE ALL LIKE ABC****
```

o

```
. RELEASE ALL LIKE A??DEF
```

Los asteriscos y las interrogaciones tienen el mismo efecto en la selección que en el comando SAVE.

Para recuperar las variable almacenadas en un fichero MEM, se usa el comando RESTORE.

```
. RESTORE FROM MEMFICH
```

o

```
. RESTORE FROM MEMFICH ADDITIVE
```

Cuando se ejecuta el comando RESTORE se borran todas las variables de memoria existentes, a menos que se use la opción ADDITIVE. Es importante recordarlo, ya que el uso de RESTORE sin tomar precauciones puede causar problemas en el desarrollo de programas. El comando RESTORE suele ser uno de los primeros que se ejecutan en un programa.

Si se usa la opción ADDITIVE, se añade el contenido del fichero MEM a la lista de variables en memoria que hay en ese momento. Cuando se ejecute un RESTORE ADDITIVE es importante asegurarse de que el número total de variables en memoria no excede de 256 (64 en dBaseII).

CONCATENACION DE CADENAS

Introduzca la siguiente lista de comandos:

```
. STORE 'Jorge' TO NOMBRE
. STORE 'Díaz' TO APELLIDO
. STORE NOMBRE + APELLIDO TO NOMBRE1
. ? NOMBRE1

Jorge Díaz

. STORE NOMBRE - APELLIDO TO NOMBRE1
. ? NOMBRE1

JorgeDíaz
```

Observe el efecto del signo menos en el segundo comando STORE, elimina los blancos del final de la entrada NOMBRE. Los símbolos más y me-

nos que hemos usado son los operadores de concatenación de cadenas.

El signo más (+) no elimina los espacios al final, mientras que el signo menos (-) si lo hace. Teclee los siguientes comandos:

```
. USE PRODUCTO
. ? TIPO_PROD + ID_PROV

CLAVO            1          * pone espacios hasta rellenar el tamaño del campo

. ? TIPO_PROD - ID_PROV

CLAVO1                      * No hay espacio entre CLAVO y 1
```

Se puede usar la concatenación de operadores para formatear la salida a impresora o pantalla, por ejemplo:

```
. USE NOMBRES
. SET PRINT ON              * Asegúrese que la impresora está conectada y encendida
```

Este comando saca por la impresora todo lo que salga por la pantalla.

```
. ? NOMBRE '   ' + APELLIDO * pone espacios entre NOMBRE y APELLIDO
```

OPERADORES ARITMETICOS

dBaseII tiene cuatro operadores aritméticos disponibles:

suma	+
resta	-
multiplicación	*
división	/

Se usan los paréntesis para agrupar operaciones en el orden en que se deben ejecutar en un cálculo. Los paréntesis se necesitan para cambiar el orden de precedencia de los operadores.

La siguiente lista de comandos ilustra el uso básico de estos operadores:

```
. STORE  5 TO A
. STORE  6 TO B
. STORE 20 TO C

. STORE A+B TO D
. ? D
     11
. STORE C-A TO E
. ? E
     15
. STORE A*C TO E
. ? E
    100
. STORE (A*C)/B TO E
. ? E
```

```
      16.6
. STORE A/B TO E
. ? E
. STORE A/2.5 TO D
. ? D
. STORE A/2.56 TO D
. ? D
```

Estos simples comandos muestran el efecto de cada operador. El uso de estos operadores no está restringido a variables en memoria. Se pueden usar valores que sean campos de bases de datos. Por ejemplo, una base de datos de pedidos puede contener campos como COSTO_UNID y CANTIDAD. El valor de un pedido de seis artículos cuyo costo unitario sea 2156 se puede obtener con:

```
. STORE COSTO_UNID*CANTIDAD TO VALOR
```

Este comando usará los valores de los campos COSTO_UNID y CANTIDAD del registro actual, para obtener la variable VALOR. Esta variable puede introducirse directamente en un campo de la base de datos con:

```
. REPLACE VALOR WITH COSTO_UNID*CANTIDAD
```

En este ejemplo hemos usado la multiplicación (*). Se pueden usar los demás operadores (+,-,/) de la misma forma, con datos almacenados en los campos del registro actual.

OPERADORES RELACIONALES

Los operadores relacionales son operadores de comparación:

<   menor que
>   mayor que
=   igual a
<> distinto de
<= menor o igual a
>= mayor o igual a

Estos operadores sirven para seleccionar registros que satisfagan una condición.

```
. USE PRODUCTO
. DISPLAY ALL FOR ID_PROV>1
```

Se listarán todos los registros en los que la identificación del proveedor sea mayor de 1 (2 y 3).

```
. DISPLAY ALL FOR ID_PROV<=2
```

Se listarán todos los registros en los que la identificación del proveedor sea menor o igual a 2 (1 y 2).

```
. DISPLAY ALL FOR ID_PROV <> 2
```

Se listarán todos los registros en los que la identificación del proveedor sea distinto de 2 (1 y 3).

Los operadores que acabamos de describir tienen una amplia variedad de usos, algunos de los cuales ya se han visto aquí. En los programas de ejemplo de los próximos capítulos se darán más aplicaciones. Los operadores relacionales y lógicos son una forma muy útil de seleccionar registros. Los operadores relacionales proporcionarán registros por comparación numérica. Los operadores lógicos se pueden usar para datos numéricos y de carácter.

FUNCIONES (dBaseII)

En cualquier lenguaje de ordenador hay ciertas funciones estándar disponibles que tiene también el dBaseII, como estas:

La función de número de registro. Permite al usuario imprimir el número de registro actual:

```
. USE PRODUCTO
. ? #
1                    * registro 1
. SKIP 3
. ? #
4                    * 3 registros más adelante
.SKIP -2
. ? #
2
```

* Marcador usado para indicar los registros que están marcados para borrado. Abra el fichero de productos y teclee Ctrl-U (que indica que el registro se va a marcar para borrar). A continuación liste el registro:

```
. USE PRODUCTO
(Teclee Ctrl-U)
. DISP
```

Aparecerá el registro así:

```
00001 *     + los campos de datos
```

El asterisco indica que se ha marcado el registro para borrar. En este punto, el registro no se ha eliminado de la base de datos. Para eliminar el registro definitivamente debe usar un comando PACK; la marca de borrado se puede cancelar tecleando Ctrl-U.

EOF Marcador de fin de fichero. Esta función asume un valor que puede ser cierto o falso. Es cierto si se ha llegado al final del fichero seleccionado; en caso contrario es falso. Es una función muy útil para terminar búsquedas que podrían dar un error si llegaran al final del fichero.

```
. DO WHILE .NOT. EOF
    Comandos              * El bucle DO WHILE continuará
. ENDDO                   * hasta llegar al final del fichero
```

! Un signo de exclamación delante de una variable o cadena de caracteres hace que el contenido se convierta en mayúsculas.

```
. STORE "abc" TO PRUEBA
. STORE !(PRUEBA) TO PRUEBA1
. ? PRUEBA
abc
. ? PRUEBA1
ABC
```

El uso de esta función en la entrada de datos facilita el mantenimiento de los datos en los registros, en un formato estándar.

```
. STORE "          " TO NOMBRE      * crea la variable NOMBRE
. @10,10 GET NOMBRE
. STORE !(NOMBRE) TO NOMBRE
. ? NOMBRE
```

Esta sección de código muestra la forma de leer de la pantalla sobre una variable y almacenarla en mayúsculas sin que intervenga el usuario.

TYPE La función de tipo de datos se usa para determinar el tipo de los datos que usa una expresión.

```
. STORE 123 TO NUM1
. STORE 'ABC' TO PRUEBA
. STORE T TO PRUEBA1
. TYPE (NUM1)
N                        * dato numérico
. TYPE (PRUEBA)
C                        * caracteres
. TYPE (PRUEBA1)
B                        * dato Booleano
```

Si el tipo de datos es indefinido, escribe una U.

INT Esta función devuelve el valor entero de una expresión o variable.

```
. ? INT(123.67)
     123
```

Usando la función INT de esta forma, desprecia simplemente todos los números a la derecha del punto decimal. Los paréntesis son necesarios porque, cualquier expresión dentro de ellos se evalúa primero, después se aplica la función INT. Si se usa un nombre de variable dentro de los paréntesis, la función INT devolverá la parte entera del valor de la variable.

```
. STORE 12.67 TO NUM1
. INT(NUM1)
     12
. STORE 139.7 TO NUM1
. INT(NUM1)
    139
```

etc.

La función INT usa el valor actual de la variable.

Para redondear por arriba o por debajo al entero más cercano, usamos el siguiente método:

```
. INT(VARIABLE+0.5)

. INT(12.1 + 0.5) --> 12
. INT(12.5 + 0.5) --> 13
. INT(12.9 + 0.5) --> 13
```

En cada caso, devuelve el valor entero más cercano al valor original.

VAL **Esta función convierte una cadena de caracteres formada por números, un signo y hasta un punto decimal, en la cantidad numérica correspondiente.**

```
. ? VAL('123') DA 123
. STORE VAL('123') TO NUM
. ? TYPE(NUM)
N                        * valor numérico
```

El operador VAL se puede usar también con un operador de subcadena:

```
. VAL ($ (CADENA))
. ? VAL($('12345',3,3))
345
```

STR **Esta función convierte un número o el contenido de una variable numérica, en una cadena.**

```
. STORE STR (123.46,6,2) TO NUM
. ? NUM1
123.46
. TYPE (NUM1)
C                        * caracteres
```

El segundo término entre paréntesis es la longitud de la cadena que se va a construir, en este caso 6. La longitud debe ser suficientemente grande para incluir todos los caracteres más el punto decimal, si lo lleva. El tercer número es la precisión decimal o el número de lugares decimales después del punto. Si no se especifica precisión, se asume que no hay punto decimal.

Un área potencial para su aplicación es para mostrar valores en la pantalla. Por ejemplo, en la base de datos PRODUCTO:

```
. STORE STR (COSTO,4,2) TO COSTO1
. STORE 'Este artículo cuesta' + COSTO1 TO PRINT
. ? PRINT
Este artículo cuesta 1150.00 * asumimos que COSTO1 sea 1150.00
```

LEN **Esta función nos indica el número de caracteres que hay en una cadena..**

```
. ? LEN ('ABCDE')
5
```

```
. STORE 'XYZ' TO PRUEBA
. ? LEN (PRUEBA)
```

NOTA: si se usa el nombre de un campo con la función LEN, retorna la longitud del campo.

La función LEN resulta útil cuando se requiere rellenar con blancos una entrada de datos.

```
DO WHILE LEN (PRUEBA) <= 5
STORE PRUEBA + ' ' TO TEST
ENDDO
```

$ **Función de subcadena. Devuelve una subcadena de una cadena especificada. $('NOMBRE',1,2) dará como resultado NO, una cadena de caracteres empezando por la posición 1 de la cadena definida y con una longitud de 2. La cadena de la que se toma la subcadena puede ser:**

1) una cadena definida, como la que hemos visto.
2) una expresión - que se convierta en una cadena
3) un nombre de variable - variable en memoria de caracteres

Por ejemplo:

```
. STORE 'ABCDEF' TO PRUEBA1
. ? $(PRUEBA1,2,3)   - usa una variable
BCD
```

@ **Esta es una función de búsqueda de una subcadena, averigua si una cadena contiene a otra. Devuelve la posición que ocupa el primer carácter de la primera cadena dentro de la segunda.**

```
. @ ('CD','ABCDEF')
```

La cadena CD aparece dentro de la cadena ABCDEF y comienza en la posición 3. Observe que esta función es diferente del operador relacional de cadenas. Este último indica cuándo una cadena es contenida dentro de otra; la búsqueda de subcadenas puede hacer lo mismo, pero también nos da la posición de comienzo de la primera cadena dentro de la segunda. También se pueden usar variables con esta función.

```
. STORE 'EXT' TO CADENA1
. STORE 'ABCDEXTSU' TO CADENA2
. STORE @ (CADENA1,CADENA2) TO POSICION
. ? POSICION
5
```

El operador de búsqueda de subcadenas se puede usar junto con la función de subcadena ($) para encontrar la primera posición y extraer de los datos una cadena de una longitud determinada, por ejemplo:

```
. STORE "JAIME JORGE JUAN DAVID ALEJANDRO" TO NOMBRES
. STORE 'JUAN' TO CADENA
. STORE @ (CADENA,NOMBRE) TO POS
```

```
. STORE (NOMBRES, POS, 4) TO MINOMBRE
. ? MINOMBRE
```

Aquí, la función @ se usa para localizar la posición de comienzo de la cadena JUAN dentro de la variable NOMBRES. Una vez que se conoce la posición de JUAN, se puede extraer de NOMBRES con el operador de subcadena.

CHR Esta función nos da el carácter ASCII equivalente de un número. Para el código de limpiar la pantalla, Ctrl+D:

```
. ? CHR(04)
```

limpiará la pantalla.

Si el código de vídeo inverso es 14, y 15 lo elimina:

```
. ? CHR(14)        Pondrá el vídeo inverso.
. ? CHR(15)        Quitará el vídeo inverso.
```

Esta función se puede usar para controlar algunos efectos en la impresora, como la impresión condensada. Es necesario comprobar en el manual de la impresora cuál es el código apropiado. Para la Epson FX-80, el código de impresión condensada es CHR(15)

```
. SET PRINTER ON
. ? CHR(15)
. ? 'Esto se escribe condensado
. ? CHR(18)
. ? 'Ahora escribe normal'
```

& Función de sustitución de macro. Cuando precede al nombre de una variable activa en memoria, el nombre es reemplazado por el contenido de la variable. La variable debe ser de caracteres. Esta función tiene varios usos, algunos de ellos se indican a continuación, otros en los ejemplos de los capítulos siguientes.

Podemos repetir un comando específico, por medio de &. Un programa que hace uso frecuente del comando DELETE RECORD:

```
. STORE 'DELETE RECORD ' TO M
```

A partir de ahora, cuando se necesite borrar un registro, en lugar de teclear DELETE RECORD se puede usar la función & de la siguiente forma:

```
. &M 6          * tiene el mismo efecto que
. DELETE RECORD 6
```

Suele ser útil almacenar en memoria nombres de bases de datos o parámetros de ficheros. Usando esta técnica, se pueden simplificar las referencias a ficheros. Por ejemplo, si un grupo de programas requieren acceso a más de una base de datos, se pueden salvar sus nombres en el fichero de memoria

como FICH1, FICH2, FICH3, etc. A continuación, para usar el fichero cuyo nombre está almacenado en la variable FICH1, se teclea el comando:

```
. USE FICH1
```

Con comandos similares podremos almacenar nombres de ficheros en FICH2 y FICH3 para su uso. Tenga en cuenta que dentro de FICH1, FICH2 y FICH3 se pueden incluir referencias a unidades.

```
FICH1 --> B:MIFICH        * MIFICH en la unidad B
FICH2 --> NFICH           * NFICH en la unidad por defecto
FICH3 --> C:OTROFICH      * OTROFICH en la unidad C
```

Si una base de datos está indexada y se usa un comando FIND frecuentemente para cambiar datos, puede ser útil el uso de &. Se usa una variable en memoria, BUSCA, para almacenar el valor actual del campo de clave del registro a buscar:

```
. USE PRODUCTO INDEX PROD1
. STORE 'MARTILLO' TO BUSCA
. FIND &BUSCA
```

La ventaja de esta función reside en el hecho de que la macro devolverá el contenido actual de la variable BUSCA. En un programa usado para localizar entradas, se puede encontrar el registro deseado introduciendo la información correcta en BUSCA.

FILE Esta función será lógicamente verdadera si existe el nombre del fichero en el disco en uso, en la unidad por defecto. Si existe un fichero llamado NOMBRE en la unidad A, y otro llamado NOMBRES1 en la unidad B:

```
. ? FILE(NOMBRE)
T                    * el fichero existe en la unidad en uso
. ? FILE(B:NOMBRES1)
T                    * el fichero existe en la unidad B
. ? FILE(B:NOMBRE)
F                    * el fichero no existe en la unidad B
```

La función FILE es particularmente útil en programas en los que se pregunta al usuario por el nombre del fichero. Con la función FILE, se puede comprobar el nombre del fichero y asegurarse de que existe en la unidad que se está usando o en una unidad especificada.

```
STORE "           " TO FNOMBRE
DO WHILE FICHERO=.F.
ERASE
@ 10,10 SAY "Introduzca el nombre del fichero"
@ 10,40 GET FNOMBRE
READ
STORE FILE(&FNOMBRE) TO FICHERO
ENDDO
```

TRIM Esta función se usa para eliminar los blancos al final de

una cadena especificada. dBASE rellena con blancos los datos introducidos en un campo, hasta completar la longitud del campo. El uso de estos datos en cualquier salida formateada puede producir resultados sorprendentes, si no se eliminan estos blancos sobrantes.

```
. USE NOMBRE
. ? 'El primer registro' + &NOMBRE + 'es el primer NOMBRE.'

El primer registro Pérez      es el primer NOMBRE.

. STORE TRIM (NOMBRE) TO NOMBRE1
. ? El primer registro' + &NOMBRE1 + 'es el primer registro.'

El primer registro Pérez es el primer NOMBRE.
```

Observe el efecto del comando TRIM en la línea impresa.

RANK Nos da el valor decimal del primer carácter de una cadena. Es similar a la función ASC disponible en BASIC.

```
. RANK ('ABCD')
65
. RANK ('GER')
71
. RANK ('XYZ')
88
```

DATE La función DATE devuelve la fecha del sistema en formato XX/XX/XX. El sistema puede poner la fecha al principio de la sesión o mediante la opción SET DATE TO. Si se pone la fecha en el arranque, se comprobará la validez por un calendario para que el formato sea correcto (formato del sistema DOS). Si se usa la opción SET DATE TO, no se comprueba, por lo que el formato puede ser MM/DD/AA o DD/MM/AA o AA/MM/DD. Si la fecha del sistema se ha puesto a 12/12/86 al arranque, entonces:

```
. ? DATE()
12/12/86
. SET DATE TO 23/01/86
. ? DATE()
23/01/86
```

Funciones del dBaseIII

Las funciones disponibles en dBaseIII se pueden clasificar en varios grupos, de la siguiente forma:

Tipo Función	Nombre	Descripción
Fecha y Hora	CDOW	Dia de la semana
	CTOD	Conversión de carácter a fecha
	CMONTH	Mes del calendario
	DATE	Fecha del sistema
	DAY	Dia del mes

	DOW	Día de la semana
	DTOC	Conversión de fecha a carácter
	MONTH	Mes del año
	TIME	Hora del sistema
	YEAR	Año
Manipulación de caracteres	&	Sustitución de macros
	AT	Búsqueda de subcadena
	LOWER	Fuerza minúsculas
	SPACE	Genera espacios en blanco
	SUBSTR	Selección de subcadena
	TRIM	Elimina blancos al final
	UPPER	Fuerza mayúsculas
Matemáticas	EXP	Exponencial (ex)
	INT	Entero
	LOG	Logaritmo
	ROUND	Redondeo
	SQRT	Raíz cuadrada
Conversión	ASC	Carácter a código ASCII
	CHR	Código ASCII a carácter
	LOWER	Mayúscula a minúscula
	STR	Numérico a carácter
	UPPER	Minúscula a mayúscula
	VAL	Carácter a numérico
Pruebas especializadas	BOF	Principio de fichero
	COL	Posición actual de columna en pantalla
	DELETED	Registro borrada
	EOF	Fin de fichero
	FILE	Existencia de un fichero
	LEN	Longitud de una cadena de caracteres
	PCOL	Posición columna de impresora
	PROW	Posición línea de impresora
	RECNO	Número de registro actual
	ROW	Posición actual de línea de pantalla
	TYPE	Valida una expresión

Esta lista de funciones contiene unas funciones nuevas y otras que han cambiado desde el dBaseII.

Funciones Nuevas

BOF Función de principio de fichero. La función se pone verdadera cuando se intenta leer pasado el primer registro lógico de la base de datos activa. Su misión es detectar una condición de fin de fichero cuando se lee la base de datos en orden inverso. Por ejemplo:

```
DO WHILE .NOT. NOMBRE="JAIME' .AND. .NOT. BOF()
SKIP -1
ENDDO
```

El comando SKIP -1 hace que el apuntador de registros se desplace por la base de datos, hacia arriba, hasta que se

llegue al principio del fichero. Si intenta subir más arriba del primer registro del fichero, la función BOF() se hace verdadera (.T.) haciendo que pare el bucle.

EXP **Esta función calcula el valor del exponencial, ej.:**

```
. ? EXP(1.000)
2.718

. STORE 1.000 TO Y
. ? EXP(Y)
2.718
```

LOG **Calcula el logaritmo en base e (logaritmo natural) de una expresión numérica, ej.:**

```
. ? LOG(2.718)
1.000
```

ROUND **Redondea una expresión numérica con el número de cifras decimales especificadas, ej.:**

```
. ? ROUND(145.678,1)
145.7
```

Como en EXP y LOG, el valor de la expresión se puede almacenar en una variable en memoria.

```
. STORE 34.673 TO X
. ? ROUND(X,2)
34.67

. ? ROUND(X,-1)
30
```

Si se usa un número negativo, la función ROUND devolverá un número redondeado al múltiplo de 10 más cercano, ej.:

-1 --> redondea a las decenas
-2 --> redondea a las centenas
-3 --> redondea a los millares
etc.

SQRT **Esta función calcula la raíz cuadrada de un número positivo, ej.:**

```
. ? SQRT(9.00)
3.00

. STORE 16 TO Y
. ? SQRT(Y)
4
```

El número de dígitos decimales del resultado es igual al del argumento suministrado.

Tenemos un nuevo grupo de funciones para manejar las variables de tipo fecha.

CTOD Esta función convierte una fecha que se ha introducido como una cadena de caracteres, en una variable de fecha. La cadena de caracteres debe estar en el formato:

MM/DD/AA ó DD/MM/AA

```
. STORE CTOD("06/12/86") TO FECHA0
. ? TYPE("FECHA0")
D                          * FECHA0 es una variable de tipo fecha

. STORE "05/23/86" TO FECHA1
. ? TYPE("FECHA1")
C                          * FECHA1 es una variable de caracteres

. STORE CTOD(FECHA1) TO FECHA2
. ? TYPE("FECHA2")
D                          * FECHA2 es una variable de tipo fecha
```

DTOC La función DTOC convierte una variable de fecha en una variable de caracteres.

```
. STORE "06/29/86" TO FECHA1
. STORE CTOD(FECHA1) TO FECHA2
. ? TYPE("FECHA2")
D
. STORE DTOC(FECHA2) TO FECHA3
. ? TYPE("FECHA3")
C
```

CTOD y DTOC son funciones complementarias.

MONTH Nos da el número que representa el mes de una variable de fecha.

```
. ? DATE()
06/23/86
. ? MONTH(DATE())
6
. STORE MONTH(DATE()) TO MES1
. ? MES1
6
. ? TYPE("MES1")
N
```

DAY Es similar a MONTH, pero devuelve el número que representa el día en una variable de datos.

```
. ? DATE()
06/23/86
. ? DAY(DATE())
23
. STORE DAY(DATE()) TO DIA1
. ? DIA1
23
. ? TYPE("DIA1")
N
```

YEAR Es similar a MONTH y DAY pero devuelve el número que representa el año en una variable de fecha.

```
. ? DATE()
06/23/86
. ? YEAR(DATE())
86
. STORE YEAR(DATE()) TO AÑO1
. ? AÑO1
86
. ? TYPE("AÑO1")
N
```

Se pueden establecer nuevos valores de año sumando una cifra al año existente y almacenando el resultado en una variable de memoria numérica.

```
. STORE YEAR(DATE())+20 TO NUEVAÑO
. ? NUEVAÑO
106
```

DOW Esta función devuelve un número en un rango de 1 a 7, representando los dias de la semana. Domingo es 1, Lunes es 2, etc.

```
. ? DOW(DATE())
4                       * Fecha del sistema 04/04/84
                        * es Miércoles
```

Para encontrar qué día va a ser dentro de 120 días:

```
. ? DOW(DATE()+120)
6                       * será viernes
```

CDOW Esta función devuelve el nombre del día de la semana, representado por un número en una variable de fecha.

```
. ? CDOW(DATE())
MARTES                  * Fecha del sistema 06/03/84

. STORE DATE()+11 TO NUEFECHA
. ? NUEFECHA
06/14/84
. ? CDOW(DATE())
SABADO
```

CMONTH Esta función nos da el nombre del mes del año representado por el número en la variable de fecha.

```
. ? CMONTH(DATE())
MARZO                   * Fecha del sistema 03/03/86
```

Para determinar el mes en el que estaremos dentro de 60 días, se necesita el siguiente comando:

```
. STORE DATE()+60 TO NUEFECHA
. ? NUEFECHA
05/02/86
```

```
.? CMONTH(DATE()+60)
MAYO
```

Funciones que se han renombrado en dBaseIII:

@	Búsqueda de subcadena	AT()
$()	Obtención de subcadenas	SUBSTR()
!()	Convertir a mayúsculas	UPPER()
RANK	Código ASCII de un carácter	ASC()
REMARK	su función la ejecutan	? y @..SAY
QUIT TO	ha sido reemplazado por	RUN
SET COLON	ha sido reemplazada por	SET DELIMITER
SET EJECT	ha sido incluida dentro de	REPORT
SET HEADING	Ha sido incluida dentro de	REPORT
SET RAW	Ha sido incluida dentro de	TRIM()
TEST()	Has sido reemplazada por	TYPE()

Funciones que han sido borradas:

RESET

SET DATE TO

Los operadores y funciones que acabamos de describir son útiles para la construcción de programas, como se verá más adelante.

Hay aún una característica importante del dBaseII que debe ser mencionada. Todos los comandos que hemos visto hasta ahora asumen que solamente hay una base de datos disponible. dBaseII tiene la facilidad de tener activo más de un fichero mediante el uso de áreas de trabajo primaria y secundaria. Los ficheros se pueden designar como primario y secundario por medio de los comandos SET PRIMARY y SET SECONDARY. Los datos de un fichero se pueden usar para actualizar los registros del otro mediante el uso de comandos como APPEND, REPLACE o cualquier otro que trasfiera datos entre ficheros. Los ficheros primario y secundario se identifican de la siguiente forma:

```
. SELECT PRIMARY
. USE PRODUCTO INDEX PRODIDX1
. SELECT SECONDARY
. USE B:ALMACEN
```

Estos comandos se suelen colocar al principio del programa, pero se pueden usar también en cualquier punto del mismo. Para cambiar del fichero primario al secundario, durante un programa, solo se requiere el comando SET PRIMARY ó SET SECONDARY.

```
. SELECT PRIMARY        * cambia el área de trabajo actual al fichero primario

. SELECT SECONDARY      * cambia el área de trabajo actual al fichero secundario
```

En muchas aplicaciones se requerirán un fichero primario y varios secundarios durante la ejecución. El cambio de los diferentes ficheros secundarios se realiza de esta forma:

```
. SELECT PRIMARY        * identifica el fichero principal
. USE PRODUCTO
. SET SECONDARY         * identifica el primer fichero secundario
```

```
. USE B:ALMACEN
```

Sección de comandos usando B:ALMACEN

```
. SELECT SECONDARY          * identifica un fichero secundario nuevo
. USE ARTICULO
```

Sección de comandos usando ARTICULO

El cambio del fichero secundario se consigue usando la opción SELECT dentro del programa, para redefinir el fichero secundario seleccionado. El fichero primario se puede redefinir también de la misma forma. Si se usan ficheros temporales en cualquier área de trabajo, se deben borrar mientras están en este área, los ficheros deben ser cerrados también mientras estén en el área de trabajo. En este punto, puede parecer que no hay muchos usos para esta facilidad; sin embargo, se proporcionan ejemplos en los capítulos siguientes.

Trabajando con múltiples ficheros en dBaseIII

dBaseIII permite al usuario tener hasta diez ficheros abiertos al mismo tiempo. Los ficheros se abren en áreas de trabajo numeradas del 1 al 10, de las que el área activa es en la que se arranca el programa. Las demás áreas se usan ejecutando el comando SELECT. Por ejemplo:

```
. SELECT 1
. USE FICHERO1
. SELECT 2
. USE FICHERO2
```

Este grupo de comandos dejará al usuario en el área de trabajo 2 con las bases de datos FICHERO1 y FICHERO2 abiertas en sus respectivas áreas de trabajo. En dBaseII era fácil referirse a los campos de un área de trabajo secundaria por medio del prefijo S, y dBaseIII usa una técnica similar. Cuando se abre una base de datos en una de las áreas de trabajo del 1 al 10, se le asigna un código de una letra o ALIAS, dependiendo del área de trabajo en la que esté abierta:

AREA DE TRABAJO	ALIAS
1	A
2	B
3	C
4	D
etc.	

Cuando se abre un fichero mediante el comando USE, se le da la opción de ponerle un ALIAS o seudónimo:

```
. USE FICHER1 ALIAS SIGFICH
```

Em el área de datos seleccionada se puede acceder a una combinación del seudónimo (ALIAS) del fichero y el nombre del campo unidos mediante los símbolos '->'. ej.

```
. SELECT 1
. USE FICHERO1 ALIAS F1
. SELECT 2
. USE FICHERO2 ALIAS F2
. SELECT 1                   * FICHERO 1 es ahora el fichero seleccionado
. ? F2->CAMPO1               * Hace que se imprima el contenido de los primeros campos de
                               la base de datos F2

. REPLACE CAMPO1 WITH F2->CAMPO2

                             * Se debe tener en cuenta que se puede usar REPLACE
                               para introducir dentro del área activa información desde
                               otras bases de datos, pero no se puede hacer al contrario.
```

Como regla general, el apuntador de registro usado en cada área de trabajo es independiente de lo que esté sucediendo en cualquiera otra área de trabajo. Hay situaciones en las que puede ser de gran ayuda el usar dos bases de datos al mismo tiempo. Un ejemplo de esto es usar abreviaturas para la entrada de datos. Los datos se introducen como códigos y un segundo fichero contiene la versión expandida del código que se usa para copiar la descripción completa sobre la pantalla o la impresora.

Consideremos un ejemplo de un sistema de facturación que usa códigos de productos cuando se introducen los datos, pero se requiere una descripción completa cuando se imprimen las facturas. El fichero FACTURAS se puede conectar al fichero DESCRIP por medio de una relación. Los dos ficheros se conectan por un campo clave común. Si se usa el campo clave, la base de datos activa debe contener el campo de la clave, y la base de datos conectada debe estar indexada por ese campo. Cuando el apuntador de registro de la base de datos activa cambie de posición, el apuntador de registro de la base de datos conectada se moverá al primer registro que coincida con el nuevo valor de la clave.

Si se usa un campo numérico, la base de datos conectada no debe estar indexada por este campo. Cuando se usan campos numéricos, el apuntador de registro en la base de datos conectada se posiciona en el número de registro que contenga el campo que coincida con el campo clave.

```
. SELECT 1
. USE FACTURAS
. SELECT 2
. USE DESCRIP INDEX CODIGO ALIAS CD
. SELECT 1
. SET RELATION TO CODIGOS INTO CD
```

Ahora, las bases de datos FACTURAS y DESCRIP están conectadas a través del campo CODIGO. Cuando se introduzcan facturas, los códigos se pueden convertir en sus descripciones tomadas de la base de datos DESCRIP, ya que está conectada a FACTURAS.

CAPITULO 4

Ficheros de comandos y programas

en dBaseII/III

Los ficheros de comandos son la forma en que se pueden reunir juntos los comandos descritos en las páginas anteriores, para realizar tareas útiles. Se pueden usar tres métodos para preparar un fichero de comandos. Estos son:

1) usando un proceso de textos
2) usando SED o ZIP
3) usando el editor de dBaseII/III con el comando:

```
MODIFY COMMAND <fichero>
```

La lista de comandos que viene a continuación forma un fichero de comandos que permite introducir datos en la base de datos CLIENTES. El fichero se ha escrito usando la opción ZIP, e incorporando la pantalla para mostrar los campos.

```
[SET TALK OFF]
[USE CLIENTES]
[APPEND BLANK]
************************************************************

NOMBRE CLIENTE #NOMBRE                NUMERO TELEFONO #TELEF

CALLE          #DIR1                  NUMERO CUENTA   #CUENTA

CIUDAD         #DIR2

PROVINCIA      #DIR3

************************************************************
```

Tal como está el fichero, permitirá introducir datos fácilmente, pero todavía es algo complicado, ya que se requeriría un comando DO ENTRY cada vez que se fueran a añadir nuevos datos al fichero. Un grupo de comandos modificados puede eliminar este problema:

```
[SET TALK OFF]
[STORE "S" TO ELECCION]
[CLEAR]
[USE CLIENTES]
[DO WHILE ELECCION="S"]
[APPEND BLANK]
************************************************************

NOMBRE CLIENTE #NOMBRE                NUMERO TELEFONO #TELEF

CALLE          #DIR1                  NUMERO CUENTA   #CUENTA
```

```
CIUDAD          #DIR2

PROVINCIA       #DIR3

**************************************************************
[@ 18,10 SAY "¿Más entradas? S\N"]
[WAIT TO ELECCION]
[ENDDO]
```

El efecto de estos comandos es bastante diferente del de los anteriores. Un bucle DO WHILE permite al operador introducir un grupo completo de registros de una vez, eliminando el tedioso trabajo de tener que llamar al fichero cada vez que se tenga que hacer una nueva entrada. Es posible incluir un mensaje en la pantalla de entrada, para permitir que se controle toda la operación desde ella. Una alternativa puede ser diseñar la pantalla separadamente y salvarla como un fichero, digamos PANT1, dando una apariencia más simple al fichero de comandos:

```
[SET TALK OFF]
[STORE "S" TO ELECCION]
[USE CLIENTES]
[DO WHILE ELECCION="S"]
     [DO PANT1]
     [WAIT TO ELECCION]
[ENDDO]
```

El comando @ 20,10 SAY "¿Más entradas? S\N" se ha incorporado dentro del fichero PANT1. La versión de este fichero de comandos, creada bajo MODIFY COMMAND ENTRADA, debe tener exactamente el mismo formato, asumiendo que el fichero PANT1 haya sido creado bajo ZIP. Se puede usar el signo /, y no se requieren los corchetes encerrando cada línea de comando. Observe que el fichero de comandos se hace mucho más efectivo cuando se usan estructuras DO WHILE e IF ELSE ENDIF, en el programa. Es posible diseñar ficheros de comandos que permitan introducir datos y mostrar los de la base de datos, así como modificar la información almacenada en la base de datos. Los ficheros de comandos son los elementos básicos de cualquier sistema de bases de datos.

Ficheros de comandos

Ya hemos visto que los ficheros de comandos son grupos de comandos individuales de dBaseII/III que, cuando se ejecutan en secuencia, realizan una tarea útil. Los ficheros de comandos pueden ser de dos tipos: los que se preparan con ZIP (ficheros de pantalla) o los que se preparan con MODIFY COMMAND <fichero>. También es posible preparar ficheros de comandos con un proceso de textos, que contengan comandos de formato como el que ya hemos visto (@ 20,20 SAY). La preparación de los ficheros de comandos con comandos de formato de pantalla requiere mucho tiempo y se debe evitar en lo posible. La ventaja del proceso de texto reside en su habilidad para cambiar textos en un fichero, formatear el texto de un fichero o hacer modificaciones globales (cambiar un símbolo o nombre en todo el fichero con un solo comando).

Los ficheros de comandos tiene diversos contenidos, pero en general

realizan una serie de acciones asociadas con la entrada, recuperación y manipulación de datos. Los ficheros de comandos son como los programas de cualquier lenguaje de ordenadores: deben tener una estructura. Cualquier programa consta de ciertos elemento básicos que debe aislar e incorporar el programador. Para decidir cómo se deben colocar los elementos del programa, se han adoptado algunos procesos estándar de diseño de programas. A continuación se describe una de estas técnicas.

El problema que vamos a trasladar a un fichero de comandos se debe especificar primero en lenguaje ordinario e identificar las partes principales. A continuación, se consideran en más detalla las partes individuales del problema, y los pasos necesarios para resolver cada uno, escritos con frases simples. El refinamiento posterior de este proceso permite al programador identificar los comandos que debe usar para conseguir los objetivos del programa.

Consideremos como ejemplo el establecimiento de una base de datos de las publicaciones disponibles para los empleados de una compañía, relacionados con el tipo de actividad de la compañía. Recuerde que será necesario acceder a la base de datos para actualizar, mostrar y borrar información.

Pruebe a listar las fases del proceso de actualización de la base de datos. Después del primer intento de preparación de los elementos del programa, mire la lista que hemos escrito y vuelva a escríbirla, intentando usar frases simples para simplificar la estructura. Después lístelo, debe estar suficientemente claro para que los refinamientos sucesivos del problema produzcan una versión en lenguaje ordinario reducido. Esta forma de especificación del problema puede ser como esta:

1) Se prepara un registro de todo el material disponible para el uso de los empleados.

2) Se deben mantener los registros, de forma que se puedan añadir las nuevas publicaciones, y borrar el material obsoleto.

3) Se deben mostrar en la pantalla los datos relacionados con peticiones específicas.

El siguiente refinamiento lo reduciría a:

una especificación de una base de datos

un programa para introducir datos

un programa para mostrar datos e imprimirlos

un programa para actualizar registros.

Especificaciones de la base de datos:

nombre del fichero
nombres de campos, tamaños y tipos
creación del fichero

Programa para introducir datos:

pantalla para la entrada de datos
rutina para entradas múltiples

Programa para mostrar datos:

pantalla para mostrar datos
rutina para búsquedas múltiples
opción de impresión

Programa para actualizar registros:

se requieren ficheros de índice
elección de entrada múltiple
elección de campos a modificar
impresión de los cambios
mostrar el registro antes de los cambios

De los requerimientos indicados se puede hacer una lista de los comandos necesarios para realizar los diferentes pasos. La lista debe reflejar su orden de uso. Ahora ya está en disposición de crear la base de datos y escribir los ficheros de comandos que ejecuten los pasos que hemos marcado. El siguiente esqueleto nos muestra la secuencia:

```
CREATE LIBROS
```

Complete el diálogo para definir los campos.

No introduzca datos aún.

Usando el ZIP (para las versiones de dBaseII de 8 bits) o dFormat (utilidad de formateo de pantallas para 16 bits suministrado con el dBaseIII) o SED (herramienta de formateo de pantallas en 16 bits usado con la versión de 16 bits de dBaseII) o con comandos de formateo de pantalla, cree el fichero que permitirá introducir datos dentro de la estructura de registros de LIBROS. Recuerde el uso de bucles IF ELSE ENDIF o DO WHILE para poder usar la pantalla varias veces. El grupo de comandos podría ser este:

```
USE LIBROS
SET TALK OFF
STORE "S" TO ELECCION
DO WHILE ELECCION="S"
    APPEND BLANK
    DO ENTRADA              * Pantalla para entrada de datos, creada usando ZIP
    @ 20,10 SAY "Teclee S para continuar, N para salir"

    WAIT ELECCION           * Permite cambiar el valor de ELECCION
ENDDO
```

Este programa nos permitirá introducir datos dentro del fichero LIBROS, y el resto de los elementos del programa se pueden escribir de la misma forma. Este programa no permite la impresión. Ahora se puede reunir la información para formar un programa de mantenimiento de la base de datos que contendrá los libros útiles.

Un segundo ejemplo puede ser una base de datos que contenga información relativa al personal empleado por una compañía. Usando el procedimiento que acabamos de ver, desarrollaremos un programa para esta aplicación. Liste las fases hasta que haya desarrollado un esqueleto del programa adecuado.

El desarrollo de ficheros de comandos es muy similar al desarrollo de programas para ordenador; la planificación es la fase más importante del proceso. Cuando se desarrolla un programa, lo más importante es incluir documentación exacta. La documentación debe dejar claro el propósito del programa, los nombres de las variables, su tipo y tamaño. Se deben identificar claramente los bloques de programa que realizan funciones específicas y explicar su función. Por último, es muy importante que se hagan y almacenen en lugar seguro copias de seguridad de los programas para evitar perder todo el trabajo en caso de accidente.

Los ficheros de comandos son el equivalente en dBaseII/III a los programas. Son grupos de instrucciones que, cuando se ejecutan en secuencia, realizan tareas determinadas, como añadir datos a una base de datos, mostrar los datos de una base de datos o manipular los datos en los registros de una o más bases de datos. El uso de estos ficheros se demuestra mejor en el contexto del procedimiento que se puede adoptar para construir la base de datos.

El paso más importante para establecer una base de datos es la fase de planificación. Es muy fácil caer en la tentación de comenzar con una vaga idea de cómo se va a completar el proyecto. Esto se debe evitar a toda costa. La planificación adecuada da como resultado un ahorro de tiempo en fases posteriores del desarrollo de aplicaciones con bases de datos.

En la fase de planificación, se deben tener siempre en cuenta los siguientes puntos:

1) ¿Está totalmente descrito el problema?

2) ¿Se conocen los resultados requeridos de la base de datos?

Para especificar los campos requeridos, se deben documentar claramente todos los aspectos del problema. Esto nos dará una indicación de los requerimientos del sistema y lo que puede ocupar. dBaseII no está muy limitado en cuanto al número de registros (el máximo es 64000). Sin embargo, es posible que la limitación de campos (32 por registro) pueda causar algún problema. Si una aplicación determinada requiere más de 32 campos por registro, la solución es usar dos bases de datos con un campo en común, por lo menos, en las dos. El campo común proporciona una conexión entre los dos ficheros, conectando los registros de una con los correspondientes en la otra. dBaseIII permite al diseñador de la aplicación hasta 128 campos por registro y mil millónes de registros en una base de datos.

Como ejemplo práctico, consideremos el problema de establecer dos bases de datos, una para contener una lista de clientes y la otra para ala lista de las compañías y su cuota de mercado asociada. La base de datos de clientes requerirá actualizaciones ocasionales, mientras que la otra requerirá actualizaciones regulares.

Base de datos de Clientes

Para la base de datos propuesta, liste los campos que usted piense que se van a necesitar:

Nø/CAMPO	NOMBRE	TAMAÑO	TIPO	DECIMAL
0001	COMP	30	C	
0002	DIRC1	20	C	
0003	DIRC2	20	C	
0004	DIRC3	20	C	
0005	CONT	30	C	
0006	TITULO	20	C	
0007	TELEF	12	N	
0008	EXT	4	N	
0009				
0010				
0011				
0012				

Recuerde que en esta fase se puede corregir un error de estructura con el comando MODIFY STRUCTURE. Ahora ya tenemos el fichero para la información de clientes, y se pueden introducir los datos por medio de uno de los métodos que ya hemos visto, como el comando APPEND o el diseño de una pantalla usando el ZIP. Considerando la operación de este fichero, decida el sistema que desea usar.

Introduzca los datos por medio del comando APPEND o de la pantalla ZIP.

La elección de la primera opción involucrará una considerable cantidad de trabajo, y requerirá un conocimiento del sistema. La segunda opción permitirá a un operador entrenado actualizar o introducir nueva información sin un conocimiento del sistema. Con la referencia de la estructura del fichero que hemos listado, diseñe la pantalla ZIP desde la que se puedan introducir los datos.

Este es el fichero de comandos, ENTRADA.CMD, para la entrada de datos a la base de datos CLIENTES. Se ha escrito usando ZIP:

```
[SET TALK OFF]
[STORE "S" TO ELECCION]
[USE CLIENTES]
[DO WHILE ELECCION="S"]
   [APPEND BLANK]                 * añade un registro en blanco para la entrada de datos
   [DO PANT1]                     * mostrar la pantalla de entrada de datos
   [ACCEPT "¿Más entradas?" TO ELECCION]
[ENDDO]
```

NOTA. Cuando use ZIP para preparar ficheros de comandos, es necesario evitar el uso del símbolo /, ya que lo usa ZIP para llamar a las pantallas de comandos.

La siguiente fase es el diseño de la pantalla que mostrará el contenido de un registro especificado. Debe ser suficientemente claro, y puede ser útil alguna forma de índice. Para indexar la base de datos CLIENTES, se debe escoger un campo de índice. En este caso, pueden ser útiles dos ficheros de índice, uno basado en el nombre de la com-

pañía y otro basado en el nombre de contacto. Se pueden usar al mismo tiempo ambos ficheros de índice, permitiendo obtener un registro particular por nombre de compañía o por nombre de contacto. Los ficheros índice se pueden crear usando el comando INDEX:

```
. USE CLIENTES
. INDEX ON COMP TO COMPIDX
. INDEX ON CONTACTO TO CONTIDX
. QUIT
```

Ya tenemos los ficheros de índice COMPIDX.IDX y CONTIDX.IDX, y se pueden usar con el comando FIND para localizar registros específicos tanto por nombre de compañía como por nombre de contacto. Se pueden hacer ambas funciones con el comando LOCATE, sin embargo, FIND es más rápido. El fichero de comandos para esta sección requerirá decidir la combinación de ficheros que se va a usar, dependiendo de la elección de la clave. En esta situación es útil usar los comandos de formateo de pantalla + y . para guardar la elección en una variable de memoria y usar después esta variable para seleccionar la combinación correcta de base de datos y fichero de índice. El siguiente fichero de comandos es una posible solución (los comentarios que aparecen en estos programas no deben de escribirse en el fichero de comandos):

Fichero de comandos SELEC

```
MODIFY COMMAND SELEC    *  SELEC es el nombre del fichero de comandos ;

ERASE                      *  limpia la pantalla
SET TALK OFF               *  elimina la salida del comando  a pantalla
STORE "S" TO REPITE        *  prepara el control del bucle DO WHILE
DO WHILE REPITE="S"        *  establece el bucle DO WHILE
   STORE " " TO ELECCION   *  prepara el control para el bucle IF THEN ELSE
     @ 5,10 SAY "  1> SELECCIONAR REGISTRO POR COMPAÑIA"
     @ 10,10 SAY "  2> SELECCIONAR REGISTRO POR CONTACTO"
     @ 15,10 SAY "  ELIJA UNA OPCION"
     WAIT TO ELECCION      *  selecciona campo de búsqueda
          IF ELECCION="1"  *  busca por nombre de compañía
              USE CLIENTES INDEX COMPIDX
              ERASE
              ACCEPT "Introduzca Nombre de Compañía" TO CNOMB
              FIND &CNOMB  *  usa una variable para el valor de la clave
              DO PANT2     *  pantalla para mostrar registro
              WAIT
          ELSE             * buscar por nombre de contacto
              USE CLIENTES INDEX CONTIDX
              ACCEPT "Introduzca Nombre de Contacto" TO CONOM
              FIND &CONOM
              DO PANT2
              WAIT
          ENDIF
     ERASE
     @ 10,10 SAY "¿Más búsquedas? S/N"
     WAIT TO REPETIR
ENDDO                      * permite volver a ejecutar el fichero
```

Este fichero de comandos ilustra una forma de usar los ficheros de índice selectivamente, para conseguir resultados lo más rápidos posibles y permitir búsquedas múltiples por cualquiera de los dos fiche-

ros de índice. Si no está disponible ninguna de estas dos piezas de información, se puede usar el comando LOCATE para buscar en el campo correspondiente a los datos disponibles.

La pantalla de muestra de datos, PANT2, se diseñará usando ZIP de forma que presente en la pantalla toda la información contenida en un registro. Puede ser como este:

```
NOMBRE COMPAÑIA     @COMP           CONTACTO  @CONT

DIRECCION           @DIRC1                    @TITULO

                    @DIRC2                    @TELEF

                    @DIRC3                    @EXT

  Pulse una tecla para continuar...
```

Para terminar el proceso de mantenimiento de la base de datos CLIENTES, escribiremos un fichero de comandos para modificar los datos en registros especificados, como se requería en la preparación. El proceso de cambiar información en un registro se hará localizando primero el registro e introduciendo los nuevos datos después. La siguiente lista de comandos puede ser una forma de hacerlo:

Fichero de comandos ACTUALIZ

```
SET TALK OFF
STORE " " TO REPITE
ERASE
   DO PANT3
   WAIT TO ELECCION
       IF ELECCION="1"
           USE CLIENTES INDEX COMPIDX
           ACCEPT " Nombre de compañía" TO NOMBRE
           ACCEPT " Nuevo nombre " TO NNOMBRE
           FIND &NOMBRE
           REPLACE COMP WITH &NNOMBRE
       ELSE
           IF ELECCION="2"
               USE CLIENTES INDEX CONTIDX
               ACCEPT " Nombre del contacto" TO VCONT
               ACCEPT " Nuevo nombre del contacto" TO NCNOMB
               FIND &VCONT
               REPLACE CONT WITH NCNOMB
           ELSE
               IF ELECCION="3"
                   USE CLIENTES
                   ACCEPT "Número de teléfono actual" TO TELENO
                   ACCEPT "Nümero de teléfono nuevo" TO NTELEF
                   LOCATE FOR TELEF="TELENO"
                   REPLACE TELEF WITH NTELEF
               ENDIF
           ENDIF
       ENDIF
```

Este programa permite cambiar el contenido de tres campos. Se puede extender para cubrir otros campos como la dirección. Una vez más cre-

aremos PANT3 usando ZIP, y será así:

SE PUEDEN CAMBIAR LAS SIGUIENTES ENTRADAS

1> NOMBRE DE,LA COMPAÑIA

2> NUMERO DE TELEFONO DE LA COMPAÑIA

3> NOMBRE DEL CONTACTO DENTRO DE LA COMPAÑIA

Elija el número del campo que desea cambiar...

Salve este fichero como PANT3.

Ahora se puede examinar la base de datos CLIENTES de varias formas diferentes y se pueden cambiar datos en registros determinados, según se necesite. Para poder realizar estas tareas, se deben recordar los nombres de todos los ficheros que se han creado. Una solución alternativa a este problema es tener un sistema completo manejado por menús. En un sistema manejado por menús, una pantalla simple permite al operador seleccionar una opción determinada. El programa de menú selecciona entonces el programa de aplicación adecuado. En el ejemplo que damos aquí, la pantalla del menú permitirá al operador realizar una de estas tres operaciones:

1) introducir datos en la base de datos

2) mostrar datos seleccionados por compañía o nombre de contacto

3) modificar los datos en registros existentes

El programa de menú que realice las funciones para el sistema que hemos creado, se puede preparar como sigue:

DISTRIBUIDORA DE PUBLICACIONES S. A.

BASE DE DATOS DE CLIENTES

OPCIONES DISPONIBLES

1> INTRODUCIR DATOS NUEVOS EN LA BASE DE DATOS

2> MOSTRAR DATOS DE REGISTROS EXISTENTES

3> MODIFICAR DATOS DE LOS REGISTROS EXISTENTES

4> SALIR DE LA BASE DE DATOS

INTRODUZCA EL NUMERO CORRESPONDIENTE A LA OPCION REQUERIDA

Este formato de pantalla se prepara usando ZIP o SED y se llamará MENU1. A continuación se escribe un fichero de comandos que llama a MENU1 y selecciona la opción apropiada. El programa selecciona después el fichero de comandos que ejecutará la opción.

Fichero de comandos MENU

```
ERASE
STORE " " TO OPCION
DO MENU1                    * MENU1 es el fichero que acabamos de crear
IF OPCION="1"
   DO ENTRADA
   DO MENU1
ELSE
   IF OPCION="2"
      DO SELEC
      DO MENU1
   ELSE
      IF OPCION="3"
         DO ACTUALIZ
         DO MENU1
      ELSE
         IF OPCION="4"
            QUIT
         ENDIF
      ENDIF
   ENDIF
ENDIF
```

Se puede usar una forma similar para establecer la base de datos de compañías y su cuota de mercado. Se decide la estructura de los campos, después se escriben los ficheros de comandos y se prepara un programa de menú que permita al usuario seleccionar la función de una lista que aparecerá en la pantalla.

Después de completar estas tareas, se puede preparar un menú para todo el sistema, permitiendo al usuario entrar en la base de datos de clientes o en la de cuotas de mercado.

El sistema de programas que hemos visto aquí se ha escrito primero para dBaseII, sin embargo, funcionará en dBaseIII con mínimas alteraciones como CLEAR en lugar de ERASE. También se debe tener en cuenta en este punto, que hay un cambio importante en la forma en que el dBaseIII trata las variables en memoria.

Consideremos el caso de un programa que se usa para llamar a varias opciones. El programa de menú se considera como primer nivel y los programas llamados desde él estarán en un nivel inferior. Cualquier variable en memoria creada en este segundo nivel de programa es liberada automáticamente cuando se ejecuta el comando RETURN. Este cambio requiere algún cuidado con la creación de variables en memoria que se quieran usar en más de un programa. Una solución simple a este problema es declarar todas las variables en memoria como PUBLIC. Con esto evitamos que se liberen cuando termina un programa determinado. Se pueden declarar como PUBLIC solamente aquellas variables que se necesiten durante la ejecución de un grupo de programas. La segunda fórmula es la más eficiente, ya que la primera ocupará más memoria de la que se requiere realmente. En la situación en que una subrutina re-

quiera usar una variable que se ha declarado como PUBLIC, se puede declarar como PRIVATE dentro de esta subrutina. Si se declara como PRIVATE, entonces se guardan los valores asignados a esa variable en los programas de nivel superior, y se restauran cuando la subrutina devuelva control, en lugar de eliminar la variable.

Se puede pasar información entre programas por medio de un comando DO modificado:

```
DO CLIENTES WITH S
```

La palabra WITH especifica una lista de parámetros que se suministran al programa CLIENTES. La primera línea del programa CLIENTES deberá contener el comando:

```
PARAMETERS ELECCION
```

La variable ELECCION tomará S como valor, y se ejecutará el programa CLIENTES. La sentencia PARAMETER asigna a las variables listadas, los valores que siguen a la palabra WITH, en el orden indicado.

Por último, debemos mencionar la opción PROCEDURE. Esta opción permite almacenar varios programas en un fichero de procedimientos y llamarlos cuando se necesiten. Por ejemplo, se pueden almacenar en un fichero de procedimientos los ficheros de pantalla usados en el ejemplo anterior y llamarlos cuando se necesiten. El fichero de procedimientos se identifica con el comando:

```
SET PROCEDURE TO (nombre fichero procedimientos)
```

Si se almacenaron las pantallas en un fichero llamado PANTALLA y se identificaron como PANT1, PANT2, etc. Para llamar a un procedimiento de pantalla concreto se deberá identificar primero el fichero de procedimientos con un comando SET PROCEDURE TO y después:

```
DO PANT1
```

nos dará la pantalla asociada con el procedimiento PANT1. El fichero de procedimientos contendrá varios juegos de comandos identificados por nombres de procedimiento y terminarán con un comando RETURN.

```
PROCEDURE PANT1
   :
   :
Comandos que forman PANT1
   :
   :
RETURN

PROCEDURE PANT2
   :
   :
Comandos que forman PANT2
   :
   :
RETURN
```

etc.

Los nombres de procedimientos pueden tener hasta 8 caracteres de longitud, y un fichero de procedimientos puede contener hasta 32 procedimientos. Solo se puede tener abierto un fichero de procedimientos en un momento dado.

CAPITULO 5

Comandos de Pantalla

Direccionamiento directo de pantalla

Un ordenador tiene dos sistemas de comunicación: su pantalla y su teclado. El teclado está diseñado por sus fabricantes, pero la presentación en pantalla la prepara el programador. En dBaseII, el diseño de pantallas se puede realizar bajo control del programa o usando ZIP o SED, unos programas de utilidad para el diseño de pantallas. Las opciones del programa, disponibles en dBaseII, consisten en un grupo de comandos, del que los más simples son los comandos @ y SAY. Por ejemplo:

```
@ 10,10 SAY "Esto es un ejemplo"
```

Este comando colocará la cadena "Esto es un ejemplo" en la pantalla, comenzando en la línea 10, columna 10. Se puede usar el mismo comando para colocar mensajes para introducir datos en variables en memoria o en un campo de una base de datos, mediante el comando adicional GET.

```
@ 10,10 SAY "¿Más datos? S/N " GET ELECCION
```

o

```
@ 10,10 SAY "¿Más datos? S/N "
@ 10,40 GET ELECCION
READ
```

La segunda forma del comando debe colocar el GET lo suficientemente lejos de la línea como para evitar que se solape con el mensaje. El comando READ debe seguir a la instrucción GET. La variable ELECCION puede ser una variable en memoria o un elemento de una base de datos, pero debe existir antes de poder completar el comando GET satisfactoriamente. Si no existe la variable, se generará un mensaje de error.

Para diseñar una pantalla muy simple, para introducir un nombre y una dirección, se hará lo siguiente:

```
ERASE
@ 5,10 SAY "Introduzca nombre"
@ 5,30 GET NOMBRE
@ 7,10 SAY "calle"
@ 7,30 GET ADD1
@ 9,10 SAY "Ciudad"
@ 9,30 GET ADD2
@ 11,10 SAY "Provincia"
@ 11,30 GET ADD3
@ 13,10 SAY "Código postal"
@ 13,30 GET PCOD
READ
```

En este ejemplo se asume que las variables direccionadas por la cláusula GET son campos en una base de datos. Si se usan variables en memoria, deben haber sido establecidas antes de ejecutar el GET:

```
STORE "          " TO NOMBRE
STORE "          " TO ADD1
```

etc.

Se puede usar el comando @ para posicionar el texto en la pantalla sin cláusulas GET adicionales:

```
@ 1,10 SAY "NOMBRES Y DIRECCIONES"
```

que proporcionarán cabeceras para la pantalla. Un ejemplo típico de un programa para esta aplicación es:

```
USE NOMBRES            * base de datos a usar
STORE "S" TO CONT      * variable de control

DO WHILE CONT="S"      * comienzo bucle entrada datos
APPEND BLANK           * añade un registro en blanco

ERASE                  * use CLEAR en dBaseIII
@ 5,10 SAY "Introduzca nombre"
@ 5,30 GET NOMBRE
@ 7,10 SAY "calle"     * comandos para mostrar en pantalla
@ 7,30 GET ADD1
@ 9,10 SAY "Ciudad"
@ 9,30 GET ADD2
@ 11,10 SAY "Provincia"
@ 11,30 GET ADD3
@ 13,10 SAY "Código postal"
@ 13,30 GET PCOD
READ

ERASE
@ 10,10 SAY "¿Más datos? S/N "
@ 10,40 GET CONT
READ                   * comandos para volver aentrar en la pantalla
ENDDO
```

En este fichero, los comandos de pantalla forman parte del fichero de comandos; es posible poner estos comandos dentro de un fichero separado, como PANTALLA, y volver a escribir el programa original así:

```
USE NOMBRES            * base de datos a usar
STORE "S" TO CONT      * variable de control
DO WHILE CONT="S"      * comienzo bucle entrada datos
APPEND BLANK           * añade un registro en blanco
DO PANTALLA            * llama al fichero de pantalla
ERASE
@ 10,10 SAY "¿Más datos? S/N "
@ 10.40 GET CONT
READ
ENDDO
```

Si el fichero de comandos PANTALLA crea alguna variable en memoria,

no pasará al primer fichero, si se está usando dBaseIII, a menos que se declaren como PUBLIC o se pase su valor como parte de una lista de parámetros.

Es posible una mejora posterior si se elimina el comando de limpiar la pantalla y se incluyen los mensajes como parte del fichero de pantalla. El fichero de pantalla quedará entonces así:

```
ERASE                   * use CLEAR en dBaseIII

@ 5,10 SAY "Introduzca nombre"
@ 5.30 GET NOMBRE
@ 7,10 SAY "calle"      * comandos para mostrar en pantalla
@ 7,30 GET ADD1
@ 9,10 SAY "Ciudad"
@ 9,30 GET ADD2
@ 11,10 SAY "Provincia"
@ 11,30 GET ADD3
@ 13,10 SAY "Código postal"
@ 13,30 GET PCOD
@ 10.10 SAY "¿Más datos? S/N "
@ 10,40 GET CONT
READ
```

El programa principal aparecerá así:

```
USE NOMBRES             * base de datos a usar
STORE "S" TO CONT       * variable de control
DO WHILE CONT="S"       * comienzo bucle entrada datos
APPEND BLANK            * añade un registro en blanco
DO PANTALLA             * llama al fichero de pantalla
ENDDO
```

Es importante recordar que un comando GET se debe referir a una variable de memoria existente o a un campo de una base de datos.

El formato que hemos diseñado se usa para introducir datos dentro de una base de datos. El procedimiento contrario se realiza usando solo comandos SAY; el fichero de pantalla se verá así:

```
ERASE                   * use CLEAR en dBaseIII

@ 5,10 SAY "Nombre"
@ 5,30 SAY NOMBRE
@ 7,10 SAY "Calle"      * comandos para mostrar en pantalla
@ 7,30 SAY ADD1
@ 9,10 SAY "Ciudad"
@ 9,30 SAY ADD2
@ 11,10 SAY "Provincia"
@ 11,30 SAY ADD3
@ 13,10 SAY "Código postal"
@ 13,30 SAY PCOD
@ 10,10 SAY "¿Más datos? S/N "
@ 10,40 SAY CONT
READ
```

Observe que se han reemplazado todos los comandos GET con comandos

SAY que mostrarán los datos desde las variables de memoria o campos de la base de datos referenciadas por los comandos SAY. La pantalla se verá así:

```
Nombre          Sr A. Pardo

Calle           Mayor, 23

Ciudad          Albacete

Provincia       Albacete

Código Postal   15036
```

Cuando se introducen datos se suelen cometer errores. Para corregir estos errores es importante reducir el número de registros inválidos que se pueden dar en una base de datos. Las soluciones a este problema consumen tiempo; sin embargo, hay algunas características que se pueden comprobar en la fase de entrada de datos, por ejemplo, si la entrada es numérica o carácter, qué longitud debe tener o si se requiere la conversión a mayúsculas. En dBaseII/III se puede realizar usando el programa o mediante cláusulas adicionales en el comando GET, como la cláusula PICTURE. El uso de la cláusula PICTURE modifica el comando GET para que acepte datos de un tipo y formato específicos.

```
@ 10,10 SAY "Introduzca la fecha "
@ 10,40 GET FECHA PICTURE "99/99/99"
READ
```

Esto fuerza al operador a incluir el símbolo / en la entrada de datos y requiere que se introduzcan solo números en la fecha; el uso del carácter 9 en la cláusula PICTURE nos lo asegura. Este ejemplo ilustra el control de las entradas numéricas, las fechas se introducirán en un campo DATE en dBaseIII. La entrada de caracteres se puede controlar mediante el uso de otro carácter de cualificación, en este caso la A. Para introducir un nombre se puede usar lo siguiente:

```
@ 10,10 SAY "Introduzca Nombre "
@ 10,40 GET NOMBRE PICTURE "AAAAAAAAAAAAAAAAAA"
READ
```

Esto fuerza la entrada a caracteres alfabéticos. El rango completo de opciones disponibles en dBaseII es:

9 o $	acepta solo números
A	acepta solo caracteres alfabéticos
X	acepta cualquier carácter
!	convierte la entrada de caracteres a mayúsculas
$	muestra el signo $ en la pantalla
*	muestra el símbolo * en la pantalla

Usando esta opciones de formato se pueden personalizar las pantallas como en el siguiente ejemplo:

```
NOMBRE                                    NUMERO DE CUENTA

    DIRECCION                             FECHA

    CALLE

    CIUDAD

    PROVINCIA

    CODIGO POSTAL


    ARTICULO      DESCRIPCION                           COSTO
```

```
@ 10,5 SAY "NOMBRE"
@ 10,20 GET NOMBRE PICTURE "AAAAAAAAAAAAAAAAAAAAAA"
@ 10,50 "NUMERO DE CUENTA "
@ 10,70 GET NCUENT PICTURE "AA/999/A9"
@ 12,10 SAY "DIRECCION"
@ 12,50 SAY "FECHA"
@ 12,70 GET FECHA PICTURE "99/99/99"
@ 14,10 SAY "CALLE"
@ 14,20 GET ADD1 PICTURE "AAAAAAAAAAAAAAAA 999"
@ 16,10 SAY "CIUDAD"
@ 16,20 GET ADD2 PICTURE "AAAAAAAAAAAAAAAAAAAA"
@ 16,10 SAY "PROVINCIA"
@ 16,20 GET ADD3 PICTURE "AAAAAAAAAAAAAAAAAAAA"
@ 18,10 SAY "CODIGO POSTAL"
@ 18,20 GET POSCOD PICTURE "99999"
@ 21,10 SAY "ARTICULO"
@ 21,25 SAY "DESCRIPCION"
@ 21,70 SAY "COSTO"
@ 22,10 GET ARTIC PICTURE "99999999"
@ 22,25 GET DESCR PICTURE "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
@ 22,70 GET COSTO PICTURE "99999"
```

La entrada de datos se fuerza ahora al tipo y formato determinados. Las cláusulas PICTURE no verifican si los datos son correctos; esto lo debe hacer el programa. Por ejemplo, si los números de cuenta están en un rango entre BB/200/C5 y DD/400/C6, la cláusula PICTURE aceptará una entrada con un número de cuenta inválido, pero que tenga el formato correcto, ej. XX/900/X3. La comprobación de las partes individuales del número de cuenta es un problema de programa.

La cláusula PICTURE en dBaseIII se ha expandido desde una plantilla simple, como en dBaseII, para incluir además funciones. Las funciones disponibles son:

C	muestra CR(crédito) después de un número positivo
X	muestra DR(débito) después de un número negativo
(	números negativos entre paréntesis
B	justifica a la izquierda los datos numéricos
Z	muestra el valor numérico cero como blancos
D	usa el formato de fecha americano (MM/DD/AA)
E	usa el formato de fecha europeo (DD/MM/AA)

A solo caracteres alfabéticos
! solo letras mayúsculas
R los literales en la plantilla no forman parte de los datos

En esta bréve descripción se puede ver que las funciones no se pueden aplicar sin asegurarse primero de que los datos son adecuados a la función, así:

C, X, (, B y Z	solo se pueden usar en datos numéricos
C, X y (	solo se pueden usar con SAY
D y E	se usan con datos de fecha, de carácter y numéricos
A, ! y R	se usan con datos de carácter

Además de estas funciones, hay unos caracteres de plantilla especiales en dBaseIII, que se usan para reconocer el tipo de datos permitidos y el formato que tomará la entrada:

9	acepta solo números
#	acepta dígitos, blancos y signos
A	acepta solo caracteres alfabéticos
L	acepta solo datos lógicos
N	acepta letras y dígitos
X	acepta cualquier carácter
!	convierte la entrada de caracteres a mayúsculas
$	muestra el signo $ en lugar de ceros delante
*	muestra el signo * en lugar de ceros delante
.	indica la posición del punto decimal
,	solo aparecerá si hay un número a su izquierda

El signo ! se puede usar con los comandos SAY y GET, ya que fuerza los datos mostrados a mayúsculas, no importa cómo se ha introducido, los convertirá en mayúsculas. $ y * solo tienen efecto con el comando SAY; el resto de los símbolos se usan con comandos GET. Si se usa una cláusula PICTURE en un GET de un número no entero, la posición del punto decimal debe ser especificado en la plantilla. La cláusula PICTURE se puede usar sin la opción de función. Sin embrago, si se usa, debe estar precedido del símbolo @ y la función y la plantilla deben estar separados por un espacio:

```
@ 10,10 SAY "Balance actual " C->BAL PICTURE "@X $$$,$$$.$$$"
```

Esta línea de comando pondrá en la pantalla el valor actual del campo BAL, desde la base de datos activa en el área de trabajo 3 (seudónimo C), en el formato $$$,$$$.$$$ y mostrará DB después (@X).

Debemos tener en cuenta dos puntos importantes:

1) la función de fecha no comprobará si se introduce una fecha válida; solo el tipo de datos DATE realiza esta comprobación.

2) se permite la combinación lógica de funciones, ej.

XB mostrará un campo numérico ajustado a la izquierda, o como una cadena de blancos si el valor es cero.

En dBaseIII, la cláusula PICTURE se usa para mostrar y para controlar la entrada de datos. En dBaseII se usa una cláusula separada para mostrar fechas:

```
@ 10,10 SAY FECHA USING "99/99/99"
```

La cláusula USING permite controlar el formato de la pantalla de una forma similar a la cláusula PICTURE. Otro ejemplo puede ser la colocación de un símbolo en una entrada numérica:

```
. STORE 123.56 TO COSTO
. @ 10.10 SAY COSTO USING "$999.99"
$123.56
```

Las opciones disponibles con la cláusula USING son:

O o #	imprime solo caracteres numéricos
A	imprime solo caracteres alfabéticos
X	imprime cualquier carácter
$	reemplaza los blancos al principio por signos $
*	reemplaza los blancos al principio por signos *

dBaseII

El uso de cláusulas opcionales en el comando @ permite formatear los datos de entrada y salida en la pantalla; se pueden usar también para formatear la salida a impresora. Un ejemplo típico puede ser el registro de un ticket, un ticket de pago o un cheque. Cada uno de estos ejemplos hacen uso de la información existente para rellenar la pantalla, por lo que solo tendrá comandos SAY. Una vez que se ha diseñado la pantalla, se puede cambiar el formato y mandar la salida a la impresora, en lugar de a la pantalla, mediante el comando SET FORMAT TO PRINT.

En modo PRINT, el dBaseII saltará una página en blanco; esto se puede evitar usando SET EJECT OFF. Mientras se imprime, el dBaseII ignora cualquier comando GET o cláusula PICTURE y también los comandos READ. Los datos que se van a imprimir deben estar por orden de línea y de columna dentro de la línea, ya que la impresora imprime la línea una columna detrás de otra y avanza una línea.

Se pueden usar formularios mayores de 24 líneas, suponiendo que al terminar la página se use el comando EJECT con la impresora seleccionada. Usando ZIP o SED es más fácil, ya que existe la opción de longitud de página, que se puede usar cuando se diseña un formulario. Más adelante, en este capítulo, veremos un ejemplo de un formulario de seguros que requiere un formato largo. La entrada de datos se controla a través de tres pantallas, pero la impresión del formulario será más fácil usando un fichero diferente que contenga el formulario completo.

El uso de los comandos @ y SAY para colocar el texto en una pantalla consume mucho tiempo cuando no se tiene disponible un generador de formatos. Cuando programe, la mejor alternativa para texto es el comando TEXT ENDTEXT. Este comando coloca en la pantalla todos los datos que hay entre TEXT y ENDTEXT, empezando en la posición actual

del cursor:

```
TEXT
       ESTE ES UN MENU PARA UNA BASE DE DATOS DE ESTUDIANTES



                    1> INTRODUCIR UN NUEVO ESTUDIANTE

                    2> BUSCAR UN ESTUDIANTE

                    3> LISTADO DE ESTUDIANTES MATRICULADOS
ENDTEXT
```

Cuando use este comando, es importante la posición del cursor para asegurarse de que el mensaje se sitúa en la posición adecuada en la pantalla. La posición del cursor se puede controlar mediante el comando @ sin comandos SAY ni GET.

```
@ 10,10
```

colocará el cursor en la línea 10, columna 10 y ejecutará la siguiente instrucción.

El método SAY y @ de introducir datos es generalmente satisfactorio, ya que, en la mayoría de los casos, ya existen las variables referidas por este comando. Sin embargo, hay ocasiones en las que es útil poder crear una variable bajo control del programa, especialmente si es una variable temporal.

Existen dos comandos adicionales para la entrada de datos, que crean la variable referida, si no existe ya. Son INPUT y ACCEPT. El comando INPUT puede llevar un mensaje incluido:

```
INPUT " ¿Más entradas? S/N " TO ELECCION
¿Más entradas? S/N :
```

La variable ELECCION se creará, si no existe anteriormente, y se establecerá el tipo de datos por los que se introduzcan, lógicos, numéricos, o de carácter. Una desventaja de este comando es la necesidad de encerrar la entrada de caracteres con comillas o paréntesis, un requerimiento que puede llevar a gran cantidad de errores en la +entrada de datos, que a su vez llevarán a errores del sistema. Por esta razón, INPUT se suele usar solamente para introducir datos numéricos. La entrada de caracteres se realiza mejor con el comando ACCEPT, ya que este comando acepta caracteres en una variable en memoria, sin comillas:

```
. ACCEPT " ¿Más entradas? S/N " TO ELECCION
¿Más entradas? S/N :
```

Este comando aceptará S o N desde el teclado sin necesidad de comillas.

Por último, un comando útil adicional que da tiempo al operador para leer la pantalla, es el comando WAIT. Este comando para la ejecución del programa hasta que se pulsa una tecla cualquiera, en respuesta al mensaje WAITING que genera este comando. El comando WAIT

puede referirse a una variable de memoria, como ACCEPT e INPUT.

```
WAIT TO ELECCION
```

La tecla pulsada se almacenará en la variable ELECCION.

En dBaseIII, los comandos ACCEPT, INPUT y WAIT eliminan los dos puntos que se generan automáticamente en dBaseII. En dBaseIII se puede usar un mensaje en el comando WAIT, de la misma forma que en INPUT y ACCEPT.

Los cambios en estos comandos los hacen más consistentes con el resto de la sintaxis del dBase. Sin embargo, hay un aspecto de los comandos ACCEPT e INPUT que causa problemas con los programas de dBaseII existentes. En dBaseII, si se tecleaba un retorno de carro (tecla ENTER) en respuesta a ACCEPT o WAIT, era posible comprobarlo como un espacio. En dBaseIII el retorno de carro se almacena como un carácter nulo. Un carácter nulo representa la ausencia de datos. La facilidad menos usual del carácter nulo es la facultad de que todo es igual a él, por lo tanto se requiere una nueva técnica para comprobar las respuestas con retorno de carro. Una posible solución sería:

```
SET TALK OFF
ACCEPT "¿RETORNO DE CARRO o S? TO TEST
IF TEST="S"
   ? "LETRA"
ENDIF
IF TEST=""
? "RETORNO DE CARRO1"
ENDIF
IF ''=TEST
? "RETORNO DE CARRO2"
ENDIF

.DO TEST
¿RETORNO DE CARRO o S? S
LETRA
RETORNO DE CARRO1

DO TEST
¿RETORNO DE CARRO O S? pulse RETORNO DE CARRO
RETORNO DE CARRO1
RETORNO DE CARRO2
```

Por lo que ""=TSET es un método posible de comprobar una variable nula.

Facilidades de Pantalla

dBaseII/III es un sistema de gestión de bases de datos en el que se pueden introducir datos en el sistema, mostrarlos en la pantalla o imprimirlos. Para que este proceso sea más fácil y más cómodo para el usuario, los datos se introducen en un diseño de pantalla reconocible por el operador.

El diseño de una pantalla para mostrar información no es una tarea fácil, pero el resultado final puede ser bueno si se recuerdan algunas reglas simples. Una pantalla se divide naturalmente en diferentes áreas, dependiendo de la forma en que se muestra el material. Una pantalla con mucha información tenderá a ocultar las áreas de entrada de datos; éstas se pueden resaltar usando vídeo inverso o parpadeante. Estos efectos pueden ser útiles pero, si se usan muy frecuentemente se hacen irritantes y fatigan al usuario.

La colocación de los mensajes en la pantalla es muy importante. Los mensajes colocados en la mitad de la pantalla son los que se leen más fácilmente. Cuanto más alejados estén los mensajes del centro, mayor fatiga causan al operador que tiene que buscar continuamente las preguntas. Del mismo modo, si los mensajes se colocan cerca de los bordes de la pantalla, pueden no ser vistos.

Los dos ejemplos que les mostramos son formas de introducir datos; el primero usando el comando APPEND y el segundo usando un fichero de comandos para diseñar la pantalla.

Comando APPEND:

```
Reg. Nø      1
NOMBRE     :                 :
DIR1       :                 :
DIR2       :                 :
DIR2       :                 :
TELEFONO   :        :
CUENTA     :          :
```

Esta pantalla no le dice mucho al usuario acerca de la información que se requiere para rellenar los campos.

Diseño de pantalla:

```
-----------------------------------------------------------------

          COMPAÑIA JCP - REGISTRO DE CLIENTES

  Nombre y Apellidos     :                 :
  Calle                  :                 :
  Ciudad                 :                 :
  Código Postal          :    :
  Teléfono               :       :
  Cuenta Nø              :          :

          ¿SON CORRECTOS LOS DETALLES? S/N : :
-----------------------------------------------------------------
```

La segunda pantalla es mucho más cómoda para el operador; proporciona series de mensajes que no requieren conocimientos del funcionamiento del dBaseII/III. El primer ejemplo muestra los nombres de los campos y asume que el operador conoce lo que se requiere en cada campo. La pantalla que se muestra en el segundo ejemplo se puede diseñar usando el ZIP, un programa de utilidad de diseño de pantallas que viene incluido en la versión de 8 bits del dBaseII. Como el dBaseIII es una versión de 16 bits, se usará dFormat para crear esta pantalla (ver Capítulo 6).

ZIP

ZIP es un programa de utilidad interactivo de diseño de pantallas que permite al ingeniero de aplicaciones generar ficheros de comandos para presentaciones en pantalla. Para usar el ZIP, teclee:

```
A>ZIP
```

Se limpiará la pantalla y aparecerá una lista de letras de opciones de comandos. Es útil imprimir esta pantalla y tenerla a mano durante las primeras sesiones. A continuación le damos la lista; Observe que cada letra de comando va precedida por el signo /.

COMANDOS ZIP

/	Prefijo de comando	/$	SALVAR el fichero escrito
/C	CENTRAR el texto	Q	SALIR al sistema
/T	ALTO de la pantalla	//	Pantalla de AYUDA
/I	Modo INSERCION	/<TAB>	Borde de la pantalla
/M	MITAD de línea	/B	BAJO de la pantalla
/A	AÑADIR línea o columna	/D	BORRAR carácter
/H	DIBUJA/BORRA línea horiz.	/K	ELIMINAR línea o columna
/N	SIGUIENTE pantalla	/V	DIBUJA/BORRA línea vertical
/F	PRIMERA pantalla	/P	pantalla ANTERIOR
/E	BORRAR fichero de trabajo	/L	ULTIMA pantalla
@SAY	VARIABLE	GET	VARIABLE
[]	Reservado para comandos de dBaseII		

Las variables dinámicas que se pueden cambiar durante la sesión ZIP, son:

!	<H>	Marcador HORIZONTAL	*	<V>	Marcador VERTICAL
80	<P>	Longitud página (23-88)	5	<T>	TAB/ESPACIADO (1-79)
40	<M>	MARGEN (0-132)			

El valor de la variable encerrada entre <> se puede cambiar en cualquier momento durante la sesión ZIP. ZIP se entiende mejor trabajando con él. Sin embargo, es posible mostrar los comandos que se usarán para producir la pantalla del segundo ejemplo.

Cuando se muestra la pantalla de ayuda, pulse RETURN para continuar. El siguiente mensaje es para preguntarle si el fichero es nuevo o viejo; si es viejo, teclee el nombre, si es nuevo, entre un nombre nuevo. La pantalla se limpia y el cursor se coloca en la esquina superior izquierda. Para dibujar los bordes se deben usar los siguientes comandos:

/H	-dibuja la línea horizontal superior
/V	-dibuja la línea vertical más alta
/B	-va a la parte inferior de la pantalla
/H	-dibuja la línea horizontal inferior
/TAB	-mueve el cursor al lado derecho de la pantalla
/T	-mueve el cursor a la parte superior
/V	-dibuja la línea vertical de la derecha
/TAB	-vuelve el cursor al lado izquierdo

Ahora debe aparecer el borde de la pantalla. Para teclear los men-

sajes, mueva el cursor a la línea requerida y colóquelo en esa línea. Para el título, tecléelo en el borde izquierdo y use el comando /C para centrarlo. Para indicar cuándo una variable debe ser leída o mostrada, se usan los símbolos # y @. Como ejemplo tomaremos una base de datos que contenga los siguientes campos:

CAMPO	NOMBRE	TIPO	TAMAÑO	DECIMAL
1	NOMBRE	C	30	
2	CUENTA	C	10	
3	DIR1	C	15	
4	DIR2	C	15	
5	CODPOS	C	10	
6	TELEFONO	C	12	

La pantalla final será:

```
***********************************************************
|              INTRODUZCA NOMBRE DEL CLIENTE              |
***********************************************************
|                                                         |
|   Nombre y Apellido     :#NOMBRE                      : |
|                                                         |
|   Nø de cuenta          :#CUENTA                      : |
***********************************************************
| DIRECCION DEL CLIENTE                                   |
|                                                         |
|   Calle                 :#DIR1                        : |
|   Ciudad                :#DIR2                        : |
|   Código postal         :#CODPOST                     : |
|                                                         |
| CONTACTO                                                |
|                                                         |
|   Nø Teléfono           :#TELEFONO                    : |
***********************************************************
```

Observe el uso del símbolo # para indicar dónde se debe leer una variable. En esta fase se debe usar el comando SALVAR para crear un fichero de comandos que pueda usar el dBaseII:

```
* PANTA1.CMD
ERASE
@ 0, 0 SAY "***********************************************************"
@ 1, 0 SAY "|              INTRODUZCA NOMBRE DEL CLIENTE              |"
@ 2, 0 SAY "***********************************************************"
@ 3, 0 SAY "|                                                         |"
@ 4, 0 SAY "|  Nombre y Apellido"
@ 4, 26 GET NOMBRE
@ 4, 59 "|"
@ 5, 0 SAY "|                                                         |"
@ 6, 0 SAY "|  Nø de cuenta"
@ 6, 26 GET CUENTA
@ 6, 59 "|"
@ 7, 0 SAY "***********************************************************"
@ 8, 0 SAY "| DIRECCION DEL CLIENTE                                   |"
@ 9, 0 SAY "|                                                         |"
@ 10, 0 SAY "|  Calle
@ 10, 26 GET DIR1
```

```
@ 10, 59 "|"
@ 11, 0 SAY "|   Ciudad"
@ 11, 26 GET DIR2
@ 11, 59 "|"
@ 12, 0 SAY "|   Código postal"
@ 12, 26 GET CODPOST
@ 13, 0 SAY "|                                                          |"
@ 14, 0 SAY "| CONTACTO                                                 |"
@ 15, 0 SAY "|                                                          |"
@ 16, 0 SAY "|   Nº Teléfono"
@ 16,26 GET TELEFONO
@ 16, 59 "|"
@ 17, 0 SAY "************************************************************"
READ
RETURN
```

ZIP añade automáticamente los comandos:

ERASE	al principio del fichero, para limpiar la pantalla
READ	Al final del fichero para leer las sentencias GET
RETURN	al final del fichero para que la ejecución vuelva al fichero que llamó al fichero de pantalla.

Cuando se teclea /S se da la opción de, crear un fichero de comandos (.CMD), de formato (.FMT) o de parar. Elija la C para crear un fichero de comandos. Después se le da la opción de cambiar el nombre y crear un fichero de impresión. Responda con N. En la línea de mensajes aparece:

Writing <fichero>.ZIP	crea el fichero que usará ZIP en ediciones posteriores.
Writing <fichero>.ZPR	fichero imprimible que imprimirá la pantalla como se ha diseñado
Writing <fichero>.CMD	fichero dBaseII usado para crear esta pantalla durante la ejecución del programa

Si tiene acceso a un ordenador y al ZIP, siga la secuencia que hemos descrito y salga después del ZIP. Cuando le salga el indicador del dispositivo, teclee Ctrl+P (activar la impresora) y después:

```
A>TYPE FICHERO.CMD
```

La impresión resultante contendrá una secuencia de comandos @ SAY y @ GET. Observe que el ZIP ha insertado automáticamente los comandos READ y RETURN al final y un comando ERASE al principio, también inserta un comando READ cada 64 GET. ZIP ha creado ya el fichero de comandos desde la imagen de la pantalla. Si se tiene que cambiar algún detalle en la pantalla, siga el mismo procedimiento:

```
A>ZIP
```

ZIP muestra la pantalla de comandos
    RETURN para la pantalla de AYUDA

OPTION para seleccionar un fichero viejo o nuevo
O para seleccionar el fichero viejo

ENTER el nombre del nuevo fichero

Aparecerá la imagen de la pantalla y podrá, usando el comando ZIP junto con las teclas de movimiento del cursor (Ctrl E, D, X y S), editar y salvar un fichero de comandos actualizado.

El uso de un fichero de comandos para generar una pantalla para la entrada de datos es solo una cara de la moneda. Se puede usar el mismo fichero de comandos para crear una pantalla para mostrar datos. Vuelva a entrar en el ZIP y llame al fichero de pantalla. Cambie los símbolos GET (#) por (@) y sálvelo como FORM2. El nombre se puede cambiar cuando aparezca el mensaje que se lo pide. La siguiente sección de código ilustra el uso de estas dos formas con la base de datos NOMBRES.DBF, indexada por el campo NOMBRE.

```
ERASE
SET TALK OFF
STORE 'N' TO COMPR
DO WHILE COMPR = 'N'
USE NOMBRES INDEX IDXNOMBRE       * Esta sección permite introducir datos
APPEND BLANK                      * para un nuevo registro con FORM1
DO FORM1
@ 20, 50 GET COMPR
ENDDO
ERASE
@ 10,10 SAY 'Introduzca el Nombre'  * Esta sección permite almacenar datos en un
@ 10,40 GET MNOMBRE                    * registro usando FORM2
FIND &MNOMBRE
DO FORM2
```

Pantallas ZIP sin @ ni

Si se van a mostrar los datos almacenados en un varios de registros, el uso de un fichero de comandos de pantalla puede ser muy lento. Cada vez que se llame el fichero de comandos, se debe volver a escribir la pantalla completa. Se puede crear una pantalla ZIP sin comandos @ ni #. Esta se usa para escribir el formulario de pantalla en el que se van a mostrar los datos. Veamos como ejemplo, el formulario creado para la entrada de datos en la base de datos de nombres. Cargue el fichero usando el ZIP, borre los comandos @ y sálvelo como FORM3.CMD. El formulario se diseña de forma que los mensajes empiecen en la misma columna, por ejemplo la 10, y estén espaciados a intervalos de dos líneas, comenzando por la línea 2. El siguiente fichero de comandos permitirá mostrar los datos de registros sucesivos en el formulario en pantalla:

```
ERASE
SET TALK OFF
DO FORM3                  * escribe el formulario en la pantalla

STORE 'N' TO COMPR        * variable de control del bucle DO WHILE
USE NOMBRES INDEX IDXNOMBRE
STORE 4 TO LINEA
```

```
STORE 26 TO POS
DO WHILE COMPR ='N' .AND. #<>0      * <>0 comprueba el fin del fichero cuando se usa índice
                                    ** ESTA COMPROBACION NO ES VALIDA EN dBaseIII
@ LINE,    POS  SAY  NOMBRE
@ LINE+2,  POS  SAY  CUENTA
@ LINE+6,  POS  SAY  DIR1
@ LINE+7,  POS  SAY  DIR2
@ LINE+8,  POS  SAY  CODPOST
@ LINE+12, POS  SAY  TELEFONO
@ LINE+16, 10   SAY  "¿Muestro otro registro? S/N"   * decide si se continúa
@ LINE+16, 37   GET  COMPR
READ
IF COMPR = 'S'
   SKIP                             * si se continúa, mover al siguiente registro
ENDIF
ENDDO
```

ZIP y comandos dBaseII

ZIP permite insertar comandos dBaseII dentro de un formulario. Las reglas que se deben seguir son:

1) El comando debe aparecer entre corchetes [].

2) Ambos corchetes deben aparecer en una misma línea o usar un punto y coma (;) para extender la línea de comando de dBaseII.

3) El comando entre corchetes debe estar separado del texto y de otros comandos.

Suponiendo que no se inserten comandos en la pantalla real, los corchetes flanqueando la línea de comando serán suficientes. Cuando una pantalla usa las 24 líneas disponibles, los comandos adicionales se deben colocar después de la pantalla. La razón para hacerlo así es que cada comando usa, por lo menos, una línea, el ZIP no distingue entre comandos dBaseII y formatos de pantalla. Por ejemplo, si el formato de pantalla tiene cinco líneas de código dBaseII antes de la primera línea horizontal, en el fichero, la primera línea horizontal estará en 5,0 en la pantalla, moviendo el resto de la pantalla hacia abajo. Por lo tanto, si es posible, el código se debe introducir después del formulario de pantalla. Si se debe introducir algún código antes del formato de pantalla, será necesario usar un editor de textos, como WordStar o MODI COMM para alterar los números de las líneas en el fichero de comandos, de forma que la imagen aparezca en la posición correcta en la pantalla. Usando el fichero de pantalla que hemos visto anteriormente para nombres y direcciones, se puede modificar para obtener un fichero de comandos para la entrada de datos:

```
[SET TALK OFF]
[STORE 'S' TO ELIGE]
[ERASE]
[USE CLIENTES]
[DO WHILE ELIGE='S']
[APPEND BLANK]
```

```
**********************************************************
|          INTRODUZCA EL NOMBRE Y LA DIRECCION           |
**********************************************************
|                                                        |
|      NOMBRE CLIENTE      :- #NOMBRE                    |
|                                                        |
|      DIRECCION                                         |
|                                                        |
|      CALLE               :- #DIR1                      |
|                                                        |
|      CIUDAD              :- #DIR2                      |
|                                                        |
|      PROVINCIA           :- #DIR3                      |
|                                                        |
|      CODIGO POSTAL       :- #CODPOS                    |
|                                                        |
|      NUMERO DE TELEFONO  :- #TELEFONO                  |
|                                                        |
|      NUMERO DE CUENTA    :- #NUMCUENT                  |
**********************************************************
[ACCEPT "¿MAS ENTRADAS? S/N " TO ELIGE]
[ENDDO]
```

El fichero de comandos generado será:

```
* PANTA2.CMD
ERASE
SET TALK OFF
STORE 'S' TO ELIGE
ERASE
USE CLIENTES
DO WHILE ELIGE='S'
APPEND BLANK
@  6, 0 SAY "**********************************************************"
@  7, 0 SAY "|          INTRODUZCA EL NOMBRE Y LA DIRECCION           |"
@  8, 0 SAY "**********************************************************"
@  9, 0 SAY "|                                                        |"
@ 10, 0 SAY "|      NOMBRE CLIENTE      :-                            |"
@ 10,30 GET NOMBRE
@ 11, 0 SAY "|                                                        |"
@ 12, 0 SAY "|      DIRECCION                                         |"
@ 13, 0 SAY "|                                                        |"
@ 14, 0 SAY "|      CALLE               :-                            |"
@ 14,30 GET DIR1
@ 15, 0 SAY "|                                                        |"
@ 16, 0 SAY "|      CIUDAD              :-                            |"
@ 16,30 GET DIR2
@ 17, 0 SAY "|                                                        |"
@ 18, 0 SAY "|      PROVINCIA           :-                            |"
@ 18,30 GET DIR3
@ 19, 0 SAY "|                                                        |"
@ 20, 0 SAY "|      CODIGO POSTAL       :-                            |"
@ 20,30 GET CODPOS
@ 21, 0 SAY "|                                                        |"
@ 22, 0 SAY "|      NUMERO DE TELEFONO  :-                            |"
@ 22,30 GET TELEFONO
@ 23, 0 SAY "|                                                        |"
@ 24, 0 SAY "|      NUMERO DE CUENTA    :-                            |"
```

```
@ 24,30 GET NUMCUENT
@ 25, 0 SAY "|                                                        |"
@ 26, 0 SAY "**********************************************************"
READ
ACCEPT "¿MAS ENTRADAS? S/N " TO ELIGE
ENDDO
RETURN
```

Si se compara este fichero con el formato anterior, podemos ver que el primer juego de coordenadas está en diferente posición en la pantalla que el primer formato. Se puede usar el WordStar o el comando MODI COMM para alterar las coordenadas de forma que estén colocadas correctamente en la pantalla.

Ficheros de pantalla para formularios largos

En muchas aplicaciones, la longitud del formulario por el que se van a introducir los datos es mayor de 24 líneas. Si se construye un formulario de más de 24 líneas, ZIP asume que es para impresora y genera un fichero imprimible. Sin embargo, se puede generar un fichero de comandos por etapas.

El formulario que se va a diseñar se divide primero en secciones de menos de 24 líneas (20 es mejor). Cada subformulario se construye y salva como un fichero de comandos. Cuando se han introducido y salvado todos los formularios en varios ficheros de comandos, vuelva al sistema. Ahora es posible construir un formulario largo por medio del comando PIP para concatenar los ficheros individuales y formar uno grande.

Para obtener un solo fichero de comandos use:

```
PIP LARGO.CMD = LARGO1.CMD, LARGO2.CMD, LARGO3.CMD
```

Donde LARGO1.CMD, LARGO2.CMD y LARGO3.CMD, son los subformularios, mientras que LARGO.CMD es el fichero completo.

De esta forma, el comando PIP unirá los tres ficheros y producirá un solo fichero de comandos, LARGO.CMD. Un buen ejemplo de este problema es un formulario de solicitud de un seguro. Los ficheros de comandos que vamos a ver a continuación se han creado usando el ZIP y unido con el comando PIP, para obtener un fichero grande que permita la entrada de datos. Los ficheros para mostrar datos se pueden crear de la misma forma, usando @ en lugar de # o, si ya existe el fichero de entrada, se puede copiar a un fichero nuevo. Después edite las sentencias # y @.

```
*** Fichero SCRPER.ZPR

        FORMULARIO DE SOLICITUD DE SEGURO PERSONAL               Pág. 1
--------------------------------------------------------------------------
NUMERO DE POLIZA  #POLNUM             NOMBRE DEL CLIENTE @MCNOMBRE

DETALLES DEL PROPIETARIO
```

```
1       DIRECCION DEL PROPIETARIO, SI NO ES SU CASA  #PDIR1
                                                     #PDIR2
2       EDAD DEL PROPIETARIO                         #PEDAD
3       SUMA ASEGURADA POR EL EDIFICIO               #SUME
4       SUMA ASEGURADA POR EL CONTENIDO              #SUMC

5       ¿HA SUFRIDO EL EDIFICIO ALGO DE ESTO?
        A) INUNDACION  B) HUNDIMIENTO  C) FALLAS O MOVIMIENTOS  S/N  #DAÑOS

6       TIPO DE RESIDENCIA
        A) CASA                  B) CHALET
        C) APARTAMENTO           D) MANSION   INTRODUZCA TIPO #RTIPO
        E) HABITACION ALQUILADA
7       ¿ESTA EN PROCESO DE CONSTRUCCION O SIN REGISTRAR?  S\N
        EN CASO AFIRMATIVO, DETALLES  #PREGIST

        ¿SON CORRECTOS LOS DETALLES?  S\N  #PRUEBA
```

Esta pantalla se usa para introducir los detalles de la propiedad No está muy bien diseñada, ya que tiene mucho texto escrito y la entrada de los datos es difícil de encontrar. Para rediseñar la pantalla tenemos que cambiar el orden en el que aparecen las preguntas y alterar el diseño de forma que los puntos de entrada de datos estén cerca del centro de la pantalla.

```
*** Fichero SCRPER.ZPR

        FORMULARIO DE SOLICITUD DE SEGURO PERSONAL                  Pág. 2
------------------------------------------------------------------------------
1  CONTENIDO   TIPO DE COVERTURA DISPONIBLE

A) INDEMNIZACION B) NUEVO por VIEJO C) NUEVO por VIEJO + DAÑOS DE ACCIDENTE
                                            INTRODUZCA EL TIPO  #CTIPO
2 UN OBJETO PARTICULAR >5% DEL VALOR DEL CONTENIDO       S\N    #PCINCO
3 VALOR TOTAL DE OBJETOS DE ORO o JOYERIA >33%           S\N    #PTREINTA

4 CUBIERTOS TODOS LOS RIESGOS      S\N
   EN CASO AFIRMATIVO, INTRODUZCA LOS DETALLES O <RETURN> PARA SALTARLO
   OBJETOS    DESCRIPCION COMPLETA (NUMEROS DE SERIE)     SUMA ASEGURADA
   #OBJ1
   #OBJ2
   #OBJ3
   #OBJ4
   #OBJ5
DINERO PERSONAL PERDIDO (SUMA ASEGURADA 100)                    #PMONEDA
USO FRAUDULENTO DE TARJETAS DE CREDITO (SUMA ASEGURADA 100) S\N #PCCARD
EQUIPO DEPORTIVO (INTRODUZCA LA SUMA ASEGURADA)                 #SMONEDA
TIPO DE DEPORTE                                                 #STYPO

         ¿ESTAN CORRECTOS LOS DETALLES?  S\N  #PRUEBA
```

Esta pantalla acepta detalles de la cobertura personal requerida. La cuarta entrada permite introducir hasta cinco objetos individuales. Cada objeto es una línea en la que se introduce el número de objeto, descripción y valor. Este método de almacenar datos no es muy satisfactorio, ya que para sacarlos se deben tratar las diferentes partes de la cadena de cada objeto. La mejor solución es usar una base de datos secundaria para la segunda parte del formulario.

```
&CABACERA
-----------------------------------------------------------------------------
CARAVANA  (RETURN para SALTARLO)
MARCA y MODELO, AÑO DE FABRICACION #PCVAN1       NUMERO DE SERIE  #PCVAN2
CARAVANA + EQUIPO ESTANDAR     (SUMA REQUERIDA)                    #PCVAN3
EFECTOS PERSONALES + EQUIPAJE  (SUMA REQUERIDA)                    #PCVAN4

SEGURO DE BICICLETAS (MARCA, DESCRIPCION, AÑO DE FABRICACION + SUMA ASEGURADA)
                                                                   #PCYCLE

ACCIDENTE PERSONAL (PONGA NOMBRE, ALTURA, PESO y FECHA DE NACIMIENTO)
                                                                   #PACCID

RIESGOS EN CARRETERA, FERROCARRIL, MAR y ATRACO (BENEFICIO REQUERIDO)
A) ASEGURADO,1 UNIDAD   B) ASEGURADO,2 UNIDADES   C) ASEGURADO+FAMILIA,1 UNIDAD
D) ASEGURADO,2 UNIDADES,FAMILIA 1 UNIDAD   INTRODUZCA BENEFICIO   #RRSA1

FRIGORIFICO  (INTRODUZCA CANTIDAD o RETURN PARA SALTARLO)          #PDEEP

FECHA ULTIMA RENOVACION #LRENEW   PERIODO DE RENOVACION  #PERIOD

FECHA PROXIMA RENOVACION  @NRENEW
```

En este fichero se toma la cabecera de una variable de memoria; es una técnica útil cuando se repite la misma cadena frecuentemente en un grupo de programas. La cadena se almacena en una variable de memoria y se escribe a un fichero en memoria al que se llama al principio de un grupo de programas.

Los ficheros que hemos visto se unen para permitir al usuario proceder a través de cada página del formulario, introduciendo los detalles relevantes de cada pregunta. Un ejemplo típico del código puede ser este:

```
DO WHILE PRUEBA="N"
DO SCRPER1
ENDDO
STORE "N" TO PRUEBA
DO WHILE PRUEBA="N"
DO SCRPER2
ENDDO
STORE "N" TO PRUEBA
DO WHILE PRUEBA="N"
DO SCRPER3
ENDDO
```

Esta sección de código muestra una forma de resolver el problema. no contiene ningún código para comprobar las respuestas ni permite al operador modificar los campos introducidos antes de seguir con otra página del formulario. Usando el comando PIP se puede crear un fichero grande, editándolo para cambiar los comandos GET en comandos SAY, y usando el ZIP se puede crear un fichero imprimible.

Ficheros de Formato

Los ficheros de formato se usan para editar pantallas formateadas, o cuando se añade un registro. Si se usa el comando EDIT, se muestra el registro con los nombres de los campos y sus contenidos. El valor de un campo se puede editar colocando el cursor en el campo y cambiando directamente el contenido. Como en la entrada de datos, la edición se puede hacer más fácil mediante el formateo de la pantalla. Para crear un fichero de formato que nos permita editar una pantalla, se puede usar el ZIP.

Usando la base de datos NOMBRES como ejemplo, se puede cargar el formulario para la entrada de datos y cambiar #NOMBRE por @NOMBRE para que el formulario no sea el mismo. Sálvelo como FORM4.FMT, seleccionando la opción FORMATO, y cambiando el nombre a FORM4. El contenido del campo NOMBRE solamente se puede mostrar, ya que es un campo de índice; esta precaución de seguridad se considera para aplicaciones prácticas. En principio se pueden editar todos los campos.

En el siguiente programa se muestra el uso de esta técnica:

```
ERASE
SET TALK OFF
STORE 'S' TO CONT
USA NOMBRES INDEX IDXNOMB
DO WHILE CONT = 'S'
    SET FORMAT
    EDIT
    @ 22, 10 SAY "¿Continúo? S/N"
    @ 22, 30 GET CONT
        IF CONT = 'S'
        @ 22, 10 SAY ''
        ENDIF
ENDDO
```

Si se ejecuta este fichero de comandos, se introducen algunos cambios dentro de los campos mostrados y se salvan los registros, cuando se vuelve a mostrar un registro, aparecerán los cambios. A continuación mostramos un fichero de formato típico preparando con ZIP. La diferencia entre los ficheros de comandos y los de formato es que los ficheros de comandos se pueden usar para preparar imágenes de pantalla así como programas de aplicación. Los ficheros de formato se usan específicamente para generar imágenes de pantalla y tiene una función más especializada.

```
*** Fichero ORDEDIT.ZPR ***

                           NUMERO DE PEDIDO  @ORD:NO
ARTICULO         DESCRIPCION                  CANTIDAD COSTO UNIDAD  TOTAL
+--+---------------------------------------+---------+----------+----------+
|1 |#ART1                                  |#CANT1   |#U:COST1  | #TOTAL1  |
|2 |#ART2                                  |#CANT2   |#U:COST2  | #TOTAL2  |
|3 |#ART3                                  |#CANT3   |#U:COST3  | #TOTAL3  |
|4 |#ART4                                  |#CANT4   |#U:COST4  | #TOTAL4  |
|5 |#ART5                                  |#CANT5   |#U:COST5  | #TOTAL5  |
|6 |#ART6                                  |#CANT6   |#U:COST6  | #TOTAL6  |
|7 |#ART7                                  |#CANT7   |#U:COST7  | #TOTAL7  |
+--+---------------------------------------+---------+----------+----------+
```

```
CTRL C  SALTAR ENTRADA ADELANTE       CTRL R   SALTAR ENTRADA ATRAS

CTRL Q  SALIR SIN CAMBIOS             CTRL W   ESCRIBIR CAMBIOS EN BASE DATOS

                <RETURN> PARA IR AL SIGUIENTE ARTICULO
```

Observe que este fichero ha incluido una lista de los comandos usados para la edición de los campos; es interesante mantener la pantalla suficientemente pequeña para permitirlo. El número de pedido se muestra aparte porque es un campo de índice; es una precaución para prevenir dificultades con los ficheros de índice. Si se requiere cambiar una variable de índice en un procedimiento de edición, se actualizará el fichero índice, suponiendo que se haya seleccionado en el comando USE o sea seleccionado por un comandos SET INDEX.

Ficheros de Pantalla usando Bases de Datos Primarias y Secundarias

En este método de operación es posible usar bases de datos primarias y secundarias para comparar datos, escribirlos o presentrarlos. A continuación se muestra un ejemplo de este tipo de pantalla; involucra dos bases de datos de proceso de pedidos.

```
*** Fichero ARTMAND.ZPR ***
                                        ** ENTRADA DE ARTICULOS RECIBIDOS **
---------------------------------+---------------------------------------
NUMERO DE PEDIDO  @ORD:NO        | NOTA SERVICIO    #NOTASERV
                                 |
PROVEEDOR         @S:CODE        | REPARTIDOR       #REPART
---------------------------------+--------------------------+------------
ARTICULOS PEDIDOS                | SERVIDOS   SI(S)   NO(N) | CANTIDAD
---------------------------------+--------------------------+------------
1 @ART1                 @CANTID1 |#DART1                    |#MANCANT1
2 @ART1                 @CANTID2 |#DART2                    |#MANCANT2
3 @ART1                 @CANTID2 |#DART2                    |#MANCANT3
4                                |                          |
5                                |                          |
6                                |                          |
7                                |                          |
-------------------------------------------------------------------------
```

Este fichero hace referencia a una variable secundaria: el código de proveedor. Naturalmente se puede extender; sin embargo el fichero de pantalla debe tener solamente un formato de pantalla. La información se debe recoger y mostrar bajo control de programa.

Mostrar Datos

El uso de la pantalla para mostrar datos de una base de datos es un recurso muy potente para las operaciones interactivas en la base de datos. Hemos visto varios tipos de ficheros diferentes, requeridos para introducir, mostrar o editar datos. La preparación de estos ficheros de comandos se puede realizar con un procesador de texto o con el ZIP, usando los comandos @ y #. No siempre es posible dar formato

a la pantalla e incluir comandos SAY y/o GET, por ejemplo, cuando se están usando dos bases de datos o cuando se tienen que escribir los datos en diferente orden del normal línea a línea. Se puede diseñar una pantalla que no contenga comandos SAY ni GET y se use solamente como imagen de pantalla en la que se escriben los datos por programa.

Este principio es aplicable tanto a la entrada de datos como a su presentación. A continuación tenemos un ejemplo:

```
*** Fichero PROFILE.ZPR ***

                    REGISTRO DE CLIENTE
------------------------------------------------------------------------
APELLIDO #APELL                         NOMBRE #NOMBRE
------------------------------------------------------------------------
DIRECCION #DIR1                | PAPELES 1> #PAPEL1
                               |
          #DIR2                |         2> #PAPEL2
                               |
          #DIR3                | BEBIDAS 1> #BEB1
                               |
TELEFONO  #CTELENO             |         2> #BEB2
                               |
OFICINA   #OTELENO             | COMIDA  1> #COM1
                               |
                               |         2> #COM2
------------------------------------------------------------------------

FORMA DE PAGO    #PAYBY              CHEQUES CAJA      #CHEQUE
ESTADO CREDITO   #CSTATUS
COMENTARIOS      #COMENT             ACEPTA CHEQUES    #ACHEQUE

¿SON CORRECTOS LOS DETALLES? <S>     <N>     #VERIF
```

En esta pantalla, la entrada de datos se hace primero por la parte izquierda hacia abajo, y después por lá parte derecha hacia abajo. Si se ha preparado la pantalla usando ZIP, los datos se leen desde la izquierda a la derecha a lo largo de cada línea. El fichero de comandos que viene a continuación muestra como se puede salvar este problema usando una pantalla sin comandos SAY ni GET y manejando los datos por programa.

```
ERASE
SET TALK OFF
USE PROFILE INDEX IDXPROSN
STORE " " TO CONT
DO WHILE CONT = " "
DO IPROFILE
APPEND BLANK
STORE "N" TO VERIF
DO WHILR VERIF="N"
@ 3, 9 GET APELL
@ 3,47 GET NOMBRE
@ 5,11 GET DIR1
@ 7,11 GET DIR2                     * el orden de los comandos @ indica
@ 9,11 GET DIR3
@ 11,11 GET CTELENO                 * el orden en que se leen los datos
@ 13,11 GET OTELENO
```

```
@ 5,44 GET PAPEL1
@ 7,44 GET PAPEL2
@ 9,44 GET BEB1
@ 11,44 GET BEB2
@ 13,44 GET COM1
@ 15,44 GET COM2
@ 18,17 GET PAYBY
@ 19,17 GET CSTATUS
@ 20,17 GET COMENT
@ 18,56 GET CHEQUE
@ 20,56 GET ACHEQUE
@ 22,46 GET VERIF
READ
ENDDO
ERASE
@ 10,10 SAY "PARA CONTINUAR  TECLEE     <RETURN>"
@ 12,10 SAY "PARA VOLVER AL MENU PULSE UNA TECLA"
@ 14,10 SAY "-----------------------------------"
@ 14,52 GET CONT
READ
ENDDO
```

El fichero de comandos usa el comando GET para aceptar datos dentro de los campos de la base de datos, en una secuencia que viene determinada por las coordenadas @, produciendo un proceso de entrada más natural. Si se requiere, se puede repetir el proceso para mostrar los datos mediante el uso de un fichero igual, pero reemplazando los comandos GET por comandos SAY. Pero si se va a imprimir esta información, los comandos SAY deben estar en secuencia de línea y columna, ya que la impresora no puede volver atrás.

Ficheros de Comandos - programas dBaseII

Los ficheros de comandos son la forma en que se han reunido los comandos en los ejemplos de las páginas anteriores, para realizar tareas útiles. Se pueden usar tres métodos para preparar un fichero de comandos:

1) usando un procesador de texto

2) usando el ZIP

3) usando el comando de dBaseII MODIFY COMMAND <fichero>

El proceso de texto es el sistema más flexible para la creación de ficheros de comandos. Con sus sofisticadas facilidades de manejo de textos, es muy fácil escribir ficheros largos y editarlos después.

ZIP permite escribir ficheros de comandos al mismo tiempo que se crea un fichero de pantalla, aunque tiene ciertas limitaciones que pueden irritar si se está escribiendo un fichero largo. ZIP tiene una longitud de página por defecto de 23, que es bastante pequeña si se diseña la pantalla al mismo tiempo que se escribe el fichero de comandos. El tamaño de página puede ser alterado, pero es irritante tener que cortar a mitad del trabajo para restaurar el tamaño de página, porque se ha pasado del espacio. Al tener ciertos caracteres re-

servados, no es posible el uso del / en las opciones S/N. La función de número de registro (#) es un carácter reservado, para la instrucción GET, en el sistema ZIP.

Finalmente, lo más importante a recordar es que todas las instrucciones de dBaseII se deben encerrar entre corchetes.

La última opción en la lista es el comando MODIFY COMMAND <fichero> del dBaseII. Esto invoca a un editor interno para escribir ficheros de comandos. En este sistema, el texto del fichero de comandos se puede teclear directamente dentro del fichero, sin restricciones en el uso del / o de # para la opción S/N o como función de número de registro. La lista de comandos que viene a continuación forma un fichero de comandos que permite introducir datos dentro del fichero CLIENTES. Se debe escribir un fichero usando la opción ZIP, en el que se incorpora la pantalla para mostrar los campos.

```
[SET TALK OFF]
[USE CLIENTES]
[APPEND BLANK]
*****************************************************************

NOMBRE CLIENTE  #NOMBRE                NUMERO TELEFONO  #TELENO

CALLE           #DIR1                  NUMERO CUENTA    #NUMCUE

CIUDAD          #DIR2

PROVINCIA       #DIR3

*****************************************************************
```

El fichero que acabamos de ver permite introducir datos fácilmente, pero es todavía un poco complicado, ya que se requiere el comando DO ENTRY cada vez que se van a añadir nuevos datos. Un grupo de comandos modificados puede salvar este problema:

```
[SET TALK OFF]
[STORE "S" TO ELIGE]
[ERASE]
[USE CLIENTES]
[DO WHILE ELIGE="S"]
[APPEND BLANK]
*****************************************************************

NOMBRE CLIENTE  #NOMBRE                NUMERO TELEFONO  #TELENO

CALLE           #DIR1                  NUMERO CUENTA    #NUMCUE

CIUDAD          #DIR2

PROVINCIA       #DIR3

*****************************************************************
[@ 18,10 SAY "¿MAS ENTRADAS? S\N"]
[WAIT TO ELIGE]
[ENDDO]
```

El efecto de estos comandos es bastante diferente del anterior. Un bucle DO WHILE permite al operador la opción de introducir una cadena completa de registros de datos de una sola vez, eliminando así el trabajo tedioso de tener que llamar al fichero cada vez que se hace una nueva entrada. Es posible incluir un mensaje en la pantalla, permitiendo controlar toda la operación desde la misma.

Una alternativa es diseñar la pantalla por separado y salvarla como el fichero PANTA1, dando una apariencia mucho más simple al fichero de comandos, así:

```
[SET TALK OFF]
[STORE "S" TO ELIGE]
[ERASE]
[USE CLIENTES]
[DO WHILE ELIGE="S"]
[DO PANTA1]
[WAIT TO ELIGE]
[ENDDO]
```

El comando @ 20,10 SAY "¿MAS ENTRADAS? S\N" se ha incorporado ahora dentro del fichero PANTA1. La versión de este fichero de comandos creado bajo MODIFY COMMAND, debe tener el mismo formato, asumiendo que el fichero PANTA1 se haya creado con ZIP. Se puede usar el signo / y no se requieren los corchetes flanqueando los comandos.

Observe que el fichero de comandos se hace mucho más efectivo cuando se usan las estructuras DO WHILE ey IF THEN ELSE. Es posible diseñar ficheros de comandos que permitan introducir y presentar datos desde una base de datos, así como permitir añadir información a un programa según se está ejecutando. Durante la ejecución del programa se pueden cambiar las constantes, permitiendo alterar la información almacenada en parte o en todos los campos de un registro.

El diseño de pantallas es una parte muy importante del diseño de aplicaciones con bases de datos. No importa lo bueno que sea el programa, si el usuario no puede comprender los datos que debe introducir. La imagen de la pantalla es una parte muy importante para hacer sentirse al usuario seguro con el sistema y no se debe dejar nada para una segunda versión.

Las técnicas ilustradas aquí se pueden usar en dBaseIII. Se debe tener cierto cuidado con el uso de las variables de memoria en los ficheros de pantalla. Si no se declaran PUBLIC estas variables, dejarán de existir cuando vuelva el control al programa que lo llama. Puede ocurrir que una variable determinada no se pueda declarar PUBLIC, pero que se pueda pasar aún el valor al programa llamado, usando la técnica de pasar parámetros.

dBaseIII tiene su propio editor de pantalla, dFormat, que puede usarse para generar diseños de pantalla y ficheros de comandos. (vea el Capítulo 6 para el dFormat).

CAPITULO 6

dFormat

dFormat es un generador de formatos de pantalla que se suministra con el dBaseIII. Este programa se ejecuta desde el sistema:

```
a>DFORMAT
```

o desde el dBaseIII con:

```
. RUN DFORMAT
```

Si su sistema tiene solamente el mínimo de memoria requerido (256K) para ejecutar el dBaseIII, el último comando no funcionará.

dFormat permite al usuario diseñar una pantalla para entrada o presentación de datos. La pantalla terminada se convierte en un fichero de formato (.FMT) de dBaseIII. Las variable de memoria o campos de base de datos se pueden mostrar (>) o introducir (<), suponiendo que el nombre del campo o variable en memoria vaya precedido por el símbolo correcto. Una vez que se ha llamado al programa, se limpia la pantalla y aparece el menú principal. Las opciones disponibles son:

1) leer el manual en pantalla
2) editar un fichero existente
3) crear un fichero nuevo
4) volver a modo edición
5) generar el código en dBaseIII
6) salir

Para el usuario que lo invoca por primera vez, la mejor opción es la número 1, el manual en pantalla. Este manual contiene resúmenes de cada sección e información detallada sobre los comandos y facilidades del dFormat.

El manual está organizado de la misma forma que un manual impreso, pero está disponible en pantalla para usarlo durante la sesión de trabajo. Las partes principales de este manual son:

PAGINA	CONTENIDO	RESUMEN
10	usando el manual	20
11	ideas DFM	21
30-34	ficheros de formato	40-44
55-60	comandos de edición	61-66

Cada página del manual puede contener varias pantallas. A medida que se ven estas pantallas, se pueden imprimir, o se puede presentar el índice en la pantalla, el usuario puede moverse hacia adelante y hacia atrás en cualquier momento. Los comandos que lo hacen son:

Comando	Acción
P	Imprimir la página que hay en pantalla
C	Ver la página de contenido
RETURN	Si ha encontrado toda la información que buscaba, RETURN le devolverá a donde estaba antes de ir al manual
Espacio	Ir a la página siguiente
B	Volver a la página anterior

Cuando comience a usar el manual, resista la tentación de imprimir el contenido completo y usarlo como si fuera un manual normal. Imprima las hojas de resumen y use el manual como una herramienta directa. No es necesario moverse por el manual de página en página, introduzca simplemente el número de la página que necesita y pulse RETURN. La siguiente pantalla será el principio de esa página. Una página puede constar de más de una pantalla; para ver las pantallas sucesivas pulse la barra de espaciado.

La versión original de 8 bit del dBaseII tenía la herramienta de formato de pantalla ZIP, que le permitía generar pantallas y producir el código dBase necesario para ejecutarla durante una aplicación. dFormat hace la misma función para la versión de 16 bit de dBaseII/III. En términos generales, se abre el fichero en el generador de formatos y se diseña la pantalla usando los comandos de movimiento del cursor para colocar los mensajes e instrucciones de las variables de los comandos GET y SAY. Se declaran o especifican máscaras donde se deben usar. Cuando se ha completado la pantalla a gusto del usuario, se salva como el fichero declarado al principio del proceso de diseño. El control vuelve a la pantalla de control de dFormat, donde se usa la opción G para para crear el código dBase necesario para recrear la pantalla durante una aplicación.

Visión general de dFormat

Por ejemplo, se requiere una pantalla sencilla para almacenar el número de matrícula del coche de un seguro. La pantalla será más o menos así:

```
----------------------------------------
----------------------------------------

  Detalles del seguro de vehículos

  Marca del vehículo        <vemarca

  Número de la matrícula   <matric

----------------------------------------
----------------------------------------
```

Para crear esta pantalla con el dFormat, use la opción de fichero nuevo. Se requiere un nombre de fichero. Después se limpiará la pantalla y aparecerá la línea de estado en lo alto de la pantalla. Esta línea de estado nos muestra el nombre del fichero que estamos usando, el modo de operación, si estamos en modo inserción o no, y nos dice que la tecla de ayuda es F1. En esta fase el modo será de no inserción, que significa que los caracteres se introducirán en la pantalla en la posición del cursor a medida que los tecleemos. La otra alternativa es el modo inserción, en el que el texto a la derecha del cursor se moverá a lo largo de la línea hacia la derecha, a medida que vayamos introduciendo caracteres. Cuando se haya completado la pantalla, se salvará pulsando F2, usando la lista de opciones para salvar el fichero.

La opción G de la pantalla de control, se usa para generar un fichero de código dBase. dFormat asumirá que el fichero de comandos se va a generar desde el fichero que se acaba de editar. Si el nombre del fichero coincide con el de uno ya existente, pulse X para cancelar la acción. A medida que dFormat crea un fichero de comandos, comprueba los errores más obvios, si encuentra alguno, se limpia la pantalla y aparecen listados los errores.

Para editar un fichero existente, use la opción E. Se usa una copia en memoria del fichero para el proceso de edición; la versión en disco permanece sin alterar hasta que se salva la versión editada. El fichero existente toma la extensión .BAK antes de que se escriba en el disco la versión editada.

Editor dFormat

Antes de poder hacer algo útil con el dFormat, será necesario familiarizarse con los comandos de control del cursor. Estos comandos se pueden dividir en grupos de acuerdo con su función.

Movimiento de caracteres y líneas

Para mover el cursor un carácter hacia la derecha o la izquierda o una línea hacia arriba o abajo, se usan la tecla de control junto con E, X, S y D, según se muestra:

↑ ^E (una línea hacia arriba)

(un carácter derecha) ^S <— —> ^D (un carácter izq.)

↓ ^X (una línea hacia abajo)

Si la pantalla que se está creando es mayor que la del ordenador, hay disponibles comandos adicionales para permitir moverse rápidamente por un fichero grande.

↑ PgUp (página anterior)

<— —>

↓ PgDn (página siguiente)

Introducción y borrado de texto

Modo no inserción (Overwrite)
En este modo, el texto se introduce en la posición del cursor; es el modo normal para el diseño de una pantalla nueva. Si se necesitan hacer cambios, coloque el cursor sobre el° error y teclee la corrección.

Modo inserción
En este modo, a medida que se teclean caracteres, el texto existente se desplaza hacia la derecha. Para cambiar de modo use la tecla Ins.

Hay dos formas de crear nuevas líneas en un fichero. Si se pulsa la tecla RETURN se produce una nueva línea. Si el cursor no está al final de la línea actual, el texto a su derecha pasará a la siguiente línea. El cursor se colocará al principio de la línea:

```
esta es una prueba del uso de RETURN para crear# una nueva línea en un fichero
```

Si el cursor está en la posición indicada por # cuando se pulsa RETURN, el resultado será:

```
esta es una prueba del uso de RETURN para crear
# una nueva línea en un fichero
```

Si el cursor está al final de la línea, se crea una nueva línea en blanco. El texto existente se moverá hacia abajo una línea para dejar sitio para la nueva.

El segundo método es usar un comando de avance de línea.

Este método hará que todo el texto a la derecha del cursor se mueva verticalmente hacia abajo junto con el cursor. Usando el ejemplo anterior, el resultado será:

```
esta es una prueba del uso de RETURN para crear
                                               # una nueva línea en un fichero
```

Borrado de texto
Para borrar texto tenemos un grupo de comandos que borran desde un carácter a una línea completa.

Top_	Borra el carácter a la derecha del cursor
Del	Borra el carácter bajo el cursor
^T	Borra la palabra completa a la derecha del cursor
^Y	Borra la línea entera desde la posición del cursor
^U	Borra la línea completa en la que está el cursor

El editor de pantalla se usa para preparar y modificar los diseños de pantalla. En el proceso es necesario salvar y/o salir de los ficheros frecuentemente o dejar el editor y volver al sistema operativo. Para salvar un fichero creado usando dFormat, pulse F2 para obtener el menú de comandos de salida. Desde este menú, use el comando S para salvar un fichero sin abandonar el editor. El comando S se debe

usar frecuentemente cuando esté editando un fichero largo, para salvar el trabajo a medida que lo ejecuta. El comando Q pasa del editor a la pantalla de control del dFormat; no se salva la versión actual del fichero. Este comando es útil si el fichero se está revisando solamente y no se tiene intención de realizar cambios o si durante el proceso de edición sucede algo que estropea la versión editada. Para salir del dFormat y volver al sistema operativo, use el comando O. El fichero actual se salva automáticamente. Si el comando usado salva el fichero editado y ya existe este fichero en el disco, se renombrará como .BAK, y el actual será un fichero .DFM. Para volver al sistema operativo sin salvar el trabajo realizado use el siguiente procedimiento:

Teclee Q para volver a la pantalla de control de dFormat
Teclee Q de nuevo para salir de dFormat

Si se han hecho cambios en el fichero actual, dFormat le pedirá confirmación de que no quiere estos cambios.

Dibujo de líneas y recuadros

Se pueden dibujar líneas y recuadros en cualquier parte de la pantalla. Para empezar, pulse F3. Le preguntará qué tipo de caracteres desea usar. Los formatos disponibles son:

líneas simples
líneas dobles
caracteres imprimibles

Para los informes debe usar caracteres imprimibles, ya que las otras opciones no imprimirán correctamente.

Para conseguir buenas definiciones de recuadros en pantalla, use las opciones de líneas. Una vez que haya elegido el tipo de caracteres, dibujará el recuadro usando las teclas de flehas. Esta teclas marcan la caja en sentido horizontal y vertical, dependiendo de la tecla que pulse. Una vez que ha completado un recuadro, con la tecla F3 la salvará a medida que la muestra en la pantalla. Se puede desactivar con la tecla F4. La tecla F4 se usa para limpiar el recuadro antes de comenzar la definición de otro con la tecla F3.

Generación de ficheros de comandos

Para generar un fichero de comandos de pantalla, se debe diseñar primero mediante el editor. La pantalla se convertirá en un fichero de comandos compuesto por una lista de sentencias con comandos GET y SAY:

```
----------------------------------------  @ 2,2 SAY "----------------------------------------"
----------------------------------------  @ 3,2 SAY "----------------------------------------"
    Detalles del seguro de vehículos      @ 5,6 SAY "Detalles del seguro de vehículos"
    Marca del vehículo        <vemarca    @ 7,6 SAY "Marca del vehículo"
                                          @ 7,31 GET VEMARCA
    Número de la matrícula    <matric     @ 9,6 SAY "Número de la matrícula"
                                          @ 9,31 GET MATRIC
----------------------------------------  @ 11,2 SAY "----------------------------------------"
----------------------------------------  @ 12,2 SAY "----------------------------------------"
```

dFormat no inserta los comandos READ y RETURN requeridos para que se ejecuten las sentencias GET y para devolver control al programa que lo llama.

El fichero de comandos generado mediante la opción G desde la pantalla de control, aparece a la derecha de la figura. Donde se ha introducido texto en la pantalla, aparecerá la línea:

```
@ LINEA,COLUMNA SAY "texto"
```

Si el se usa el símbolo de entrada de datos (<) delante de un texto, se generará el correspondiente comando GET:

```
@ LINEA,COLUMNA GET VEMARCA
```

y cuando se usa el símbolo para mostrar el contenido de una variable (>), se genera una línea:

```
@ LINEA,COLUMNA SAY <variable>
```

Las excepciones a estas reglas son:

si la línea comienza por	*	línea de comentarios
	!	una definición de formato PICTURE
	>	un campo de salida

el texto entre [] se interpreta como comandos de dBaseIII

El diseño de pantalla que vemos a continuación se usa como fichero fuente para la generación de in fichero de comandos.

```
----------------------------------------
----------------------------------------

 Detalles del seguro de vehículos

 Marca del vehículo        <vemarca

 Número de la matrícula   <matric

----------------------------------------
----------------------------------------
```

Cuando se ejecute, la pantalla se verá así:

```
----------------------------------------
----------------------------------------

 Detalles del seguro de vehículos

 Marca del vehículo       :            :

 Número de la matrícula   :        :

----------------------------------------
----------------------------------------
```

y aparecerán signos de dos puntos (:) para indicar el tamaño del campo que se ha de introducir. Tenga en cuenta que se pueden usar variables de memoria como destino de los comandos GET.

Cuando se diseña una pantalla de entrada de datos, es normal usar campos de una base de datos y variables de memoria como argumentos para los comandos GET y SAY. Es perfectamente aceptable hacer que el dBaseIII muestre el resultado de cálculos en una línea, junto con un texto explicativo, por ejemplo:

```
VALOR = >PRECIO*NUMERO
```

Este comando será válido, suponiendo que PRECIO y NUMERO existan como variables en memoria o como campos de una base de datos.

Formatos PICTURE

En el capítulo anterior se mostró cómo, el uso de la cláusula PICTURE con los comandos SAY y GET pueden ayudar al control de los datos que se presentan o introducen. dFormat permite al usuario incorporar estas cláusulas dentro de un diseño de pantalla. Un formato PICTURE puede declararse como una cláusula NONAME o como formato de nombres. Por ejemplo, la pan alla anterior se puede modificar para asegurarse que la marca del coche se introduce como caracteres y que el número de matrícula se introduce en formato de letras y caracteres:

```
!a aaaaaaaaaaaa
!b aa 9999 aa
----------------------------------------
----------------------------------------

  Detalles del seguro de vehículos

  Marca del vehículo        <vemarca!a

  Número de la matrícula    <matric!b

----------------------------------------
----------------------------------------
```

La especificación de formatos se identifica mediante un signo ! y una letra. Para usar un formato PICTURE definido, se pone el signo ! seguido por la letra, en la parte correcta de la pantalla, y dFormat generará los comandos para ejecutar la opción PICTURE. Se puede definir un formato NONAME usando solo !. Un formato PICTURE se identifica por:

!	primer carácter de la definición de formato
C	un nombre adecuado para el formato o un espacio
Formato	cualquier formato dBaseIII permitido

Ya que el dBaseIII usa la cláusula PICTURE tanto para entrada como para presentación de datos, los formatos definidos en la pantalla se pueden usar tanto con comandos SAY como con GET.

Para introducir comandos dBaseIII en el diseño de pantalla, se incluyen entre corchetes []. Cuando dFormat lee estas líneas, genera líneas de comandos dBaseIII usando el texto que aparece entre los corchetes como comandos a ejecutar:

```
* Esta pantalla acepta datos de los vehículos
[USE COCHE] [APPEND BLANK]
[CLEAR]
!a aaaaaaaaaaaa
!b aa 9999 aa
----------------------------------------
----------------------------------------

  Detalles del seguro de vehículos

  Marca del vehículo       <vemarca!a

  Número de la matrícula   <matric!b

----------------------------------------
----------------------------------------
[READ] [RETURN]
```

Cuando dFormat genere el fichero de comandos desde esta pantalla, los comandos encerrados entre corchetes se ejecutarán en secuencia:

```
* Esta pantalla acepta datos de los vehículos
USE COCHE
APPEND BLANK
CLEAR
@ 2,2 SAY "----------------------------------------"
@ 3,2 SAY "----------------------------------------"
@ 5,6 SAY "Detalles del seguro de vehículos"
@ 7,6 SAY "Marca del vehículo"
@ 7,31 GET VEMARCA PICTURE 'AAAAAAAAAAAA'
@ 9,6 SAY "Número de la matrícula"
@ 9,31 GET MATRIC PICTURE 'AA 9999 AA'
@ 11,2 SAY "----------------------------------------"
@ 12,2 SAY "----------------------------------------"
READ
RETURN
```

Los comandos se colocan automáticamente en secuencia y se calculan los números de línea y columna.

Convenciones usadas en este manual

La tecla CONTROL se representa por "^" en vez de escribir Ctrl.

La tecla de ESCAPE se representa por ESC.

Para acceder al manual mientras se está editando, pulse F1.

Los comentarios que aparecen a continuación de los comandos, precedidos por un *,son solamente informativos; si se teclean producirán errores.

Algunas cosas sobre dFormat

dFormat no reserva más de 40K para el formulario del usuario

El máximo número de líneas que se pueden usar en un formulario se calcula mediante la fórmula:

Nº de líneas = memoria reservada / ancho del formulario +1

Cuando se salva un fichero, se comprimen los tabuladores, excepto si van entre comillas.

Cuando se llama al fichero, los tabuladores se expanden.

No se deben usar caracteres nulos en un fichero que se va a editar con dFormat.

dFormat busca el manual en pantalla en el directorio seleccionado y después en el directorio \DO\.

dFormat no sabe si se le ha llamado desde una unidad diferente de la que se está usando, por lo tanto, no sabe si el fichero DFM.MSG, el manual en pantalla, está en otra unidad.

CAPITULO 7

dBaseII/III y Otras

Aplicaciones

Uso de Procesos de Texto

En muchas aplicaciones resulta de gran utilidad leer datos de una base de datos para pasarlos a un proceso de texto. Las listas estándar de correo y la preparación de etiquetas son dos ejemplos típicos.

Las cartas estándar son una facilidad común en muchas actividades de negocios. Muchos procesos de texto tienen la capacidad de mezclar datos de diferentes ficheros para construir cartas estándar. Una combinación típica de programas para realizar esta función es el dBaseII/III y el WordStar. dBaseII/III tiene la capacidad de copiar datos desde una base de datos a un fichero, usando los delimitadores especificados en el comando COPY. Podemos verlo usando la base de datos de nombres y direcciones:

```
. USE NOMBRES
. COPY TO FICH1 NOMBRE,APELLIDO,DIR1,DIR2,DIR3, ;
DELIMITED WITH ,
```

Al fichero resultante le llamaremos FICH1.TXT; la extensión TXT se añade automáticamente. En este fichero, cada campo de datos está separado de otro mediante una coma, un formato que puede ser leído por la mayoría de los procesos de texto, incluido el WordStar.

La siguiente fase de la operación es preparar el documento al que vamos a añadir los datos. Un ejemplo es una carta de instrucciones para los candidatos a un examen. El fichero de datos usado se construye desde la base de datos de las entradas confirmadas, que contiene los nombres de los candidatos. Usando el código dado, se crea el fichero de datos del proceso de texto que se usará para producir el número de copias requerido. Este proceso se puede programar como una opción en un sistema de manejo de registros manejado por menú. Sin embargo, los pasos que describimos a continuación se pueden introducir desde el teclado:

```
. USE NOMBRES
. COPY TO FICH1 NOMBRE,APELLIDO,DIR1,DIR2,DIR3, ;
DELIMITED WITH ,
```

Al final de este paso, salga al proceso de texto:

Desde dBaseII:

```
. QUIT to 'WS'     * en CP/M de 8 bits
```

Desde dBaseIII:

```
. RUN WS           * asume que WS está en el dispositivo actual
```

De la lista de opciones del proceso de texto, seleccione la opción de mezclar datos desde un fichero de texto.

A continuación puede ver un documento típico de WordStar que puede aceptar datos de este tipo. Observe los comandos que permiten introducir datos donde los requiere el documento.

```
.. file cjnginst.gen
.MT 3
.LH10
.OP
.PO35
.SV FECHA-HOY, 8 Setiembre,
.DF FICH1
.RV Nombre, Dir-1, Dir-2, Dir-3, Dir-4, Fech-exam,
          Categori

N:REF: EXAM/HD

&Fecha-hoy& 1986

$Nombre&
&Dir-1&
&Dir-2&
&Dir-3&
&Dir-4&

Estimado &Nombre&,

                    TITULO DEL EXAMEN - FISICA

  Le confirmamos que participará en el examen reseñado, en este Ins-
tituto, comenzando el &Fecha-exam& de 1986.

Categoría: &categori&

  Sírvase presentarse en la sala de exámenes en el edificio B2, calle
Prieto Abad, Madrid a las 8.45.

  Los candidatos que no hayan satisfecho la matrícula deberán hacerla
efectiva antes de entrar a la sala de exámenes, ya sea en metálico o
con un cheque coformado por un banco.

  Todos los candidatos para los que se haya reservado habitación se-
rán responsables del pago de sus cuentas antes de abandonar el hotel
o lugar de hospedaje y de la notificación de cualquier cambio o can-
celación.

Atentamente

MØ Rosa Fernández. Secretaria de examen

PD. Si necesitara repetir el examen, asegúrese de mandar su solicitud
con dos meses de antelación a la FECHA DE EXAMEN.
```

Los comandos de punto al principio del fichero identifican las condiciones que se ponen en la impresora así como la fecha que se va a imprimir y el fichero de datos que se va a usar. El último comando de punto (.rv) define las variables que se van a suministrar desde el fichero de datos; estos se identifican por el signo & en el texto de la carta. &nombre& tomará el primer campo de datos y lo imprimirá en la posición de la variable &nombre&.

Para imprimir un grupo de etiquetas se sigue un sistema similar, excepto que el fichero del proceso de texto se prepara para imprimir etiquetas de acuerdo al formato de las etiquetas suministradas.

```
.DM     Este pograma producirá una lista de correo en
.DM     formato de tres líneas. Los datos deben ser así
.DM
.DM <reg #>,<nombre>,<dirección>,<ciudad>,<provincia>,<cp>,<abierto>,<cerrado>
.DM
.AV "Nombre del fichero de datos? ",ficherodatos
.DF &ficherodatos&
.FI ETIQUETA.FMT
```

Fichero ETIQUETA.FMT referido por el comando .FI anterior

```
.PL 6
.MT 0
.MB 0
.PT OFF
.OP
.PO 0
.CW 10
.RV numreg1,nombre1,dir1,ciudad1,prov1,cp1,saludo,cierre
.RV numreg2,nombre2,dir2,ciudad2,prov2,cp2,saludo,cierre
.RV numreg1,nombre3,dir3,ciudad3,prov3,cp3,saludo,cierre
.RP
&nombre1&                  &nombre2&                 &nombre3&

&dir1&                     &dir2&                    &dir3&


&ciudad1&,&prov1&,&cp1&    &ciudad2&,&prov2&,&cp2&   &ciudad3&,&prov3&,&cp3&


.CS
.DM IMPRIMIENDO REGISTRO #&numreg1&: &nombre1&
.DM IMPRIMIENDO REGISTRO #&numreg2&: &nombre2&
.DM IMPRIMIENDO REGISTRO #&numreg3&: &nombre3&
.PA
```

En la práctica hay varios métodos disponibles para combinar estas operaciones. En CP/M de 8 bits, el uso de la opción QUIT TO desde el dBaseII hace que la trasferencia del dBaseII sea muy simple:

```
. USE NOMBRES
. COPY TO FICH1 NOMBRE,APELLIDO,DIR1,DIR2,DIR3, ;
DELIMITED WITH ,
```

Desde el dBaseII:

```
. QUIT TO 'WS'              * en CP/M de 8 bit
```

Desde el dBaseIII:

```
. RUN WS                    * asume que WS está en el dispositivo actual
```

El nombre del fichero para la opción QUIT TO debe ir entre comillas simples y se puede usar un nombre de dispositivo:

```
. QUIT TO 'B:WS'
```

En dBaseIII no se requieren las comillas, pero se puede usar la especificación de dispositivo

```
. RUN B:WS
```

Además, usando el WordStar se puede usar la opción de ejecutar otro programa, permitiendo ejecutar un fichero de comandos de dBaseII/III para preparar el fichero. Al finalizar, pasará control de nuevo al WordStar.

dBaseII/III y DataStar

DataStar es un producto que permite al usuario diseñar pantallas muy complicadas para la entrada de datos, pero no tiene la misma flexibilidad que el dBaseII para manipular datos. Si se examinan los ficheros de datos del DataStar, los campos de datos van separados por comas, y por lo tanto podemos leer estos ficheros con el dBaseII.

Hay dos usos posibles para este tipo de combinación: se pueden trasferir datos desde el DataStar al dBaseII o crear ficheros en dBaseII para manipularlos con el DataStar. En el primer caso el formato de los datos de entrada se diseña en el DataStar sin estructura de campos, después se define una base de datos en dBaseII con sus correspondientes estructuras. Para trasferir los datos desde el DataStar:

```
. USE FICH2
. APPEND FROM FICH3 DELIMITED
```

Esta forma del comando APPEND lee los datos del FICH3 eliminando los delimitadores (comas) y almacenando los datos en los campos de la base de datos.

Para leer datos desde dBaseII/III dentro del formato DataStar, primero hay que copiarlos al formato delimitado por comas que se usa después como fichero de datos para el formato adecuado.

```
. USE FICH2
. COPY TO FICH3 CAMP1,CAMP2,,,DELIMITED
```

Una vez en DataStar, FICH3 se usa como fichero de datos. Observe que el comando COPY se podría haber usado con delimitadores de coma. Sin delimitadores de coma, los campos de datos van entre comillas;

con ellos, las comillas se eliminan.

Como ejemplo podemos crear bajo DataStar un fichero de datos de proveedores requerido por el sistema de gestión dBaseII. El fichero DataStar contiene nombres de proveedores, direcciones y número de teléfono, así que la base de datos debe contener campos similares. La estructura de la base de datos será:

CAMPO	NOMBRE	TIPO	TAMAÑO	DECIMALES
001	NOMBRE	C	30	
002	APELLIDO	C	30	
003	DIR1	C	30	
004	DIR2	C	25	
005	DIR3	C	25	
006	TELEFONO	C	15	

Una vez que se ha creado este fichero, los datos se pueden leer usando el comando APPEND:

```
. USE FICH4
. APPEND FROM FICH3 DELIMITED
```

dBaseII/III puede usar ahora FICH4 para manipular los datos de los proveedores.

dBaseII y el Código Ensamblador

dBaseII tiene varios comandos diseñados para poder ejecutar rutinas en ensamblador:

```
SET CALL TO <dirección>
```

La dirección a que se refiere es el equivalente decimal a una dirección hexadecimal de memoria. La dirección así referida es usada por el dBaseII en el comando CALL:

```
CALL
```

CALL ejecutará una llamada a una rutina en código máquina situada en la dirección asignada por el comando SET CALL TO. Las instrucciones almacenadas en esta dirección de memoria se asume que están en un fichero HEX, en formato HEX de INTEL, que se puede preparar de dos formas:

1 LOAD <fichero>

El comando LOAD carga el fichero dentro de la memoria con el origen en la dirección especificada en la sentencia ORG dentro del fichero. Los ficheros cargados con este comando se pueden preparar usando un programa ensamblador.

2 POKE

El comando POKE requiere una dirección seguida por una lista de datos que se almacenan en posiciones sucesivas de memoria, comenzando

por la dirección inicial:

```
POKE 41984 , 42 , 6 , 0 , 34
```

Este comando colocará los datos hex en posiciones sucesivas de memoria, comenzando en OA400H.

Se puede examinar el contenido de una posición de memoria mediante el comando PEEK, que permite al usuario ver el contenido de la posición de memoria especificada:

```
PEEK <dirección>
```

Si tiene una aplicación que entra automáticamente en dBaseII desde un fichero AUTOEXEC.BAT y no tiene disponible un reloj de tiempo real, la fecha del DOS permanecerá como valor por defecto, 1/1/80. Los ficheros creados tendrán esta fecha por defecto. Esta subrutina asume que la fecha del sistema del dBaseII se ha introducido correctamente.

PCDATE.PRG

Esta rutina está basada en la siguiente rutina en ensamblador de 8086:

```
MOV     CX,[F00F]     ; MOVER AÑO A CX
MOV     DX,[F00D]     ; MOVER MES Y DIA A CL
MOV     AH,2BH        ; NUMERO DE FUNCIOND DEL DOS EN AH
INT     21H           ; INTERRUPCION TIPO 21
MOV     [F021],AL     ; AL=0 OK, FF MAL
RET                   ; VOLVER
```

Esta rutina devuelve IS:SET = .T. si el DOS tomó la fecha
IS:SET = .F. fecha no válida para el DOS

Coloque los datos del sistema dBaseII en variables de memoria:

```
STORE VAL($(DATE(),1,2)) TO T:MES
STORE VAL($(DATE(),4,2) TO T:DIA
```

El año se almacena como dos octetos, primero el de mayor valor y después el de menos valor:

```
STORE INT(1900/256) TO Y:AÑO:H
STORE VAL($(DATE(),7,2))+1900-INT(1900/256)+256;
TO T:AÑO:L
```

Ahora se colocan en memoria de forma que la subrutina las pueda leer:

```
POKE 61453 , T:DIA , T:MES , T:AÑO:H , T:AÑO:L
```

Coloque ahora en la memoria la rutina en ensamblador:

```
POKE 61457 , 139 , 14 , 15 , 240 , 139 ; 22 , 13 , 240 ,;
             180 , 43 , 205 , 33 , 162 , 33 , 240 , 195
SET CALL TO 61457
CALL
```

Compruebe si el DOS acepta la fecha, ej. ¿está la posición F021 a 00 o a FF Hex? Usa una variable de memoria para pasar la información al programa:

```
. STORE PEEK(61473)=0 TO IS:SET
. RELEASE ALL LIKE T:*
. RETURN
```

A continuación le mostramos un ejemplo de programación usando estos comandos y que está disponible en el disco de distribución del dBaseII:

```
* Programa.: DATESYS.CMD
* Autor....: Luis A. Castro.
* Fecha....: 06/12/82.
* Observac.: Copyright 1982,  ASHTON-TATE.
* Notas....: Para poner la fecha del sistema con una fecha verificada
*      usando la subrutina DATEST. Este fichero de comandos funciona
*      solamente con dBaseII versión 2.3B y 2.4 bajo CP/M 2.2. Puede
*      eliminar todas las líneas de comentarios para que se ejecute
*      más rápido.
*
* Subrutina DATETEST:
*   Memoria usada.: A410H a A482H.
*   Descripción...: Comprueba la fecha introducida a través de POKEs
*      las posiciones 41997, 41998 y 41999. La posición 41997 tiene el
*      parámetro del mes, la 41998 tiene el del día y la 41999 la del
*      año.  Devuelve un caracteres nulos en las posiciones 41997,
*      41998 y 41999 si hay un error en la entrada. Esta subrutina
*      comprueba también los años bisiestos.
*   Nota..........: Esta subrutina estará siempre disponible para los
*      comandos POKE y CALL, una vez que se ha cargado, siempre que
*      no ejecute un comando SORT.
*
*   Este programa es solamente un ejemplo. El autor no garantiza que
*      sea sintácticamente correcto o completo. El autor recomienda
*      que el usuario mantenga siempre copias de seguridad de los
*      ficheros, especialmente cuando pruebe un nuevo programa.
*      Cualquier corrupción de los datos como resultado del uso de
*      este programa es de total responsabilidad del usuario.
*
SET TALK OFF
SET BELL OFF
SET INTENSITY OFF

STORE DATE() TO mdate
IF mdate="00/00/00"
   * Las secuencias de POKEs que siguen, cargan la subrutina
   * DATETEST comenzando en la dirección 42000 decimal. He
   * elegido el método de los POKEs porque, con una pequeña
   * subrutina es más rápido de cargar.
   SET CALL TO 41000
   *             0   1   2   3   4   5   6   7   8   9
   POKE 42000,  58, 14,164,254,  1,218,115,164,254, 32
   POKE 42010, 210,115,164, 58, 15,164,254,  0,218,115
   POKE 42020, 164,254,100,210,115,164, 58, 13,164,254
   POKE 42030,   1,218,115,164,254, 13,210,115,164,254
   POKE 42040,   2,202, 92,164, 14,  4, 33,127,164,190
```

```
POKE 42050,  35,202, 83,164, 13,194, 65,164, 58, 14
POKE 42060, 164,254, 32,210,115,164,201, 58, 14,164
POKE 42070, 254, 31,210,115,164,201, 58, 15,164,230
POKE 42080,   3, 58, 14,164,202,109,164,254, 29,210
POKE 42090, 115,164,201,254, 30,210,115,164,201, 62
POKE 42100,   0, 50, 13,164, 50, 14,164, 50, 15,164
POKE 42110, 201,  4,  6,  9, 11

* A continuación aparece un mensaje en la pantalla para
* tomar la fecha, y el programa se repite hasta que se
* introduce una fecha correcta.
ERASE
@ 2, 0 SAY 'F E C H A   D E L   S I S T E M A'
@ 3, 0 SAY '========================================'
@ 3,40 SAY '========================================'
* Inicializa los parámetros de la fecha
POKE 41997,0,0,0

DO WHILE PEEK(41997)=0
     STORE "  /  /  " TO mdate
     @ 5,0 SAY 'Introduzca fecha del sistema DD/MM/AA ';
          GET mdate PICTURE "99/99/99"
     READ
     * Ahora hace el POKE del día, mes y año dentro de
     * las posiciones 41997,41998 y 41999, respectivamente.
     POKE 41997,VAL($(mdate,4,2))
     POKE 41998,VAL($(mdate,1,2))
     POKE 41999,VAL($(mdate,7,2))
     CALL
ENDDO

* Si prefiere introducir el formato MM/DD/AA puede sustituir
* el DO WHILE...ENDDO anterior por
* DO WHILE PEEK(41997)=0
*    STORE "  /  /  " TO mdate
*    @ 5,0 SAY 'Introduzca fecha del sistema MM/DD/AA ';
*         GET mdate PICTURE "99/99/99"
*    READ
*    POKE 41997,VAL($(mdate,1,2))
*    POKE 41998,VAL($(mdate,4,2))
*    POKE 41999,VAL($(mdate,7,2))
*    CALL
* ENDDO

* En este punto se pone la fecha del sistema
SET DATE TO &mdate
ENDIF

RETURN
* EOF DATESYS.CMD
```

Hay muchas posibilidades en el área de interconexión con otros programas. Es de particular interés la conexión entre alguna hoja de cálculo y el dBaseII/III. Uno de los más notables en este área es el LOTUS 1/2/3, una hoja de cálculo muy potente que tiene limitadas las posibilidades de bases de datos. Este programa es capaz de leer ficheros dBaseII/III y usar los datos para actividades de la hoja de cálculo, incluyendo la representación de material gráfico.

CAPITULO 8

Programas dBaseII

El propósito de este capítulo es proporcionar varios ejemplos del uso del dBaseII, para ilustrar las facilidades que hemos descrito en los capítulos anteriores.

El primer ejemplo consta de dos programas de un sistema de gestión de un hotel; uno es un programa para el servicio de habitaciones y el otro es un sistema para preparar las facturas. El programa de servicio de habitaciones se puede usar directamente como está; sin embargo, el de preparación de facturas requiere la base de datos de servicio de habitaciones para completar la factura.

Los ejemplos que damos en este capítulo son solamente como ilustración y no es nuestra intención que se usen en su formato para aplicaciones practicas.

Gestión de un hotel

Aparecerá un menú que mostrará algunas de las opciones útiles en un sistema para la gestión de un hotel. Se incluye la opción de servicio de habitaciones y el sistema de preparación de facturas. Seleccionando la opción correcta, aparecerá la pantalla adecuada.

MENU DESDE EL QUE SE REALIZA LA SELECCION

```
     SELECCIONE LA OPCION DESEADA
     ----------------------------
+-------------------------+-------------------------+-------------------------+
|      RESERVAS           | PREPARAR FACTURAS       |    MANT. DE FICHEROS    |
+-------------------------+-------------------------+-------------------------+
|                         |                         |                         |
|  A1) INTRODUCIR RESERVA |  C1) PREPARAR FACTURA   | E1)                     |
|  A2) CANCELAR RESERVA   |  C2) SALIR              | E2)                     |
|  A3) SALIR              |                         | E3)                     |
|                         |                         |                         |
+-------------------------+-------------------------+-------------------------+
| SERVICIO HABITACIONES   |  ARCHIVO DE CLIENTES    |   PAGINAS DE AYUDA      |
+-------------------------+-------------------------+-------------------------+
|                         |                         |                         |
|  B1) ENTRADA DIARIA     | D1) INTRO. CLIEN. NUEVO | H1) RESERVAS            |
|  B2) RESUMEN HABITACION | D2) BORRAR CLIENTE      | H2) ARCHIVO CLIENTES    |
|  B3) RESUMEN POR DIA    | D3) VER ENTRADA CLIENTE | H3) SERVICIO HABITACION |
|                         |                         |                         |
+-------------------------+-------------------------+-------------------------+
   *** INTRODUZCA UNA OPCION
```

El fichero de comandos que muestra este menú se verá más adelante.

```
SET TALK OFF
STORE "S" TO MENTER
STORE " " TO MACTION
STORE "1" TO MRFLAG1
STORE "2" TO MRFLAG2
STORE "N" TO FLAGCANL
STORE "  " TO OPCION
* BMENU1.PRG
ERASE
@  1, 6 SAY "SELECCIONE LA OPCION DESEADA"
@  2, 6 SAY "----------------------------"
@  3, 1 SAY "+-------------------------+-------------------------+--"
@  3,56 SAY "-----------------------+"
@  4, 1 SAY "|      RESERVAS           | PREPARAR FACTURAS       |"
@  4,58 SAY "MANT. DE FICHEROS    |"
@  5, 1 SAY "+-------------------------+-------------------------+--"
@  5,56 SAY "-----------------------+"
@  6, 1 SAY "|                         |                         |"
@  6,79 SAY "|"
@  7, 1 SAY "|  A1) INTRODUCIR RESERVA |  C1) PREPARAR FACTURA   |"
IF FLAGCANL="S"
@  7,55 SAY "E1) LISTA DE CLIENTES   |"
ELSE
@  7,55 SAY "E1)                     |"
ENDIF
@  8, 1 SAY "|  A2) CANCELAR RESERVA   |  C2) SALIR              | E"
@  8,56 SAY "2)                    |"
@  9, 1 SAY "|  A3) SALIR              |                         | E"
@  9,56 SAY "3)                    |"
@ 10, 1 SAY "|                         |                         |"
@ 10,79 SAY "|"
@ 11, 1 SAY "+-------------------------+-------------------------+--"
@ 11,56 SAY "-----------------------+"
@ 12, 1 SAY "| SERVICIO HABITACIONES   |   ARCHIVO DE CLIENTES   |"
@ 12,57 SAY "PAGINAS DE AYUDA     |"
@ 13, 1 SAY "+-------------------------+-------------------------+--"
@ 13,56 SAY "-----------------------+"
@ 14, 1 SAY "|                         |                         |"
@ 14,79 SAY "|"
@ 15, 1 SAY "|  B1) ENTRADA DIARIA     | D1) INTRO. CLIEN. NUEVO | H"
@ 15,56 SAY "1) RESERVAS           |"
@ 16, 1 SAY "|  B2) RESUMEN HABITACION | D2) BORRAR CLIENTE      | H"
@ 16,56 SAY "2) ARCHIVO CLIENTES   |"
@ 17, 1 SAY "|  B3) RESUMEN POR DIA    | D3) VER ENTRADA CLIENTE | H"
@ 17,56 SAY "3) SERVICIO HABITACION |"
@ 18, 1 SAY "|                         |                         |"
@ 18,79 SAY "|"
@ 19, 1 SAY "+-------------------------+-------------------------+--"
@ 19,56 SAY "-----------------------+"
@ 20, 4 SAY "*** INTRODUZCA UNA OPCION"
@ 20,31 GET OPCION
READ
```

Observe el uso del IF ELSE ENDIF dentro de los comandos de pantalla para controlar la opción E1 del menú.

NOTA: En los siguientes ejemplos elimine los comentarios después de los comandos. Si no lo hace obtendrá un error.

```
STORE OPCION TO MROPCION
    DO CASE
        CASE OPCION="A1"
            DO TESTBOOK             * Programa reservas
        CASE OPCION="A2"
            DO TESTCANL             * Cancelar reserva
        CASE OPCION="A3"
            DO QUERY                * Buscar habitación vacía
        CASE OPCION="B1"
            DO RSERVICE             * Programa servicio habitaciones
        CASE OPCION="B2"
            DO TOTRSERV             * Costo total del servicio
        CASE OPCION="C1"
            DO INVOICE              * Preparar factura
        CASE OPCION="C2"
            ERASE                   * Salir del sistema
            QUIT
        CASE OPCION="D1"
            DO PROFILE              * Archivo clientes
        CASE OPCION="E1"

            USE B:CLIENTES INDEX B:IDXNOMB
            ERASE
            @ 10,10 SAY "ACTUALIZANDO LISTA CLIENTES ---- ESPERE POR FAVOR"
            PACK
            @ 12,10 SAY "TERMINADO    MENU PRINCIPAL ----        <RETURN>"
            WAIT
                                * Mantenimiento lista clientes
        CASE OPCION="H1"
            DO HLPTEXT1         * Páginas de ayuda
    ENDCASE
ENDDO
QUIT
```

Este fichero se presenta sin comentario, ya que forma parte de un sistema grande que es demasiado largo para incluirlo aquí. Una facilidad que hemos utilizado es el uso de señalizadores puestos por otros programas para poner mensajes en ciertas opciones del menú.

En este caso, las opciones de mantenimiento de ficheros no aparecen en el menú si no se necesitan para su ejecución. El objetivo es recalcar la necesidad de ejecutarlos en las rutinas de fin de día; para obtener mayor efecto se pueden mostrar en vídeo inverso. El señalizador que hemos usado aquí es FLAGCANL. Si tiene el valor "S", aparecerá el mensaje "LISTA DE CLIENTES", indicando que la base de datos de clientes requiere mantenimiento. FLAGCANL se pone desde el programa de preparación de facturas e indica que un cliente ha abandonado el hotel, por lo tanto se debe borrar de la lista actual de clientes.

La pantalla que presentamos a continuación se usa para introducir detalles de la factura de los clientes. Se ha preparado usando el ZIP y tiene espacio para diez entradas de datos. Para terminar, se introduce una Q en la entrada, que no se imprime en la factura. Después de que se han introducido todas las entradas, el programa busca en la base de datos los cargos por el servicio de habitaciones y los introduce automáticamente en la factura, manteniendo los totales con y sin IVA incluido.

***** Fichero BILLSCR.ZPR *****

```
  NOMBRE @NOMBRE                              HABITACION @MROOM
                                              DIAS       @MLBOOK

+---------+-----------------------------------------+---------+---------+
| CARGO1  |#ITEM1                                   | #COST1  |         |
| CARGO2  |#ITEM2                                   | #COST1  |         |
| CARGO3  |#ITEM3                                   | #COST3  |         |
| CARGO4  |#ITEM4                                   | #COST4  |         |
| CARGO5  |#ITEM5                                   | #COST5  |         |
| CARGO6  |#ITEM6                                   | #COST6  |         |
| CARGO7  |#ITEM7                                   | #COST7  |         |
| CARGO8  |#ITEM8                                   | #COST8  |         |
| CARGO9  |#ITEM9                                   | #COST9  |         |
| CARGO10 |#ITEM10                                  | #COST10 |         |
+---------+-----------------------------------------+---------+---------+
                                                    | @TOTAL  | @IVA    |
                                                    +---------+---------+
                                                              | @FTOTAL |
                                                              +---------+
ERASE
SET BELL OFF
SET TALK OFF

* En este sistema el fichero primario es la base de datos de clientes. Esto
* permite acceder a los detalles de las habitaciones almacenados en los
* registros de clientes en el momento en que se hace la reserva, y que se
* necesita para usarlo con el fichero de servicio de habitaciones.

SELECT PRIMARY
USE B:CLIENTES INDEX B:IDXNOMB
STORE "N" TO VERIFI                       ** Variable en memoria para control
STORE "                              " TO MAPELLID
STORE "                              " TO MCNOMBRE

*  Se usan variables en memoria para formar el índice por el que se va a
*  encontrar la entrada en el fichero de clientes.

DO WHILE VERIFI="N" .AND. #<>0
    @ 10,10 SAY "Introduzca apellido  "
    @ 12,10 SAY "Introduzca nombre    "
    @ 10,33 GET MAPELLID
    @ 12,33 GET MCNOMBRE
    READ
    STORE MAPELLID+$(MCNOMBRE,1,1) TO MNOMBRE
    STORE TRIM(MAPELLID)+" "+$(MCNOMBRE,1,1) TO MNOMBRE1

    * Se usa la función TRIM para eliminar los blancos posteriores

    @ 15,10 SAY "¿Es correcto el nombre?"
    @ 15,35 SAY MNOMBRE1
    @ 16,35 SAY "************************"
    @ 15,70 GET VERIFI

* Introduciendo el nombre del cliente y formando la clave para el FIND.
* Observe la opción para volver a introducir el nombre
```

```
    FIND &MNOMBRE

*  El contador de registros será 0 cuando se llegue al EOF usando un fichero
*  de índice. Para evitar un error de EOF, se usa la condición IF que vemos a
*  continuación para atrapar esta situación.

        IF #=0
             ERASE
             @ 10,10 SAY "No hay entrada por este nombre"
             @ 12,10 SAY "Para volver a introducirlo teclee <S>"
             @ 14,10 SAY "Para volver al menú teclee        <N>"
             @ 15,10 SAY "************************************"
             @ 15,30 GET VERIFI
             READ
        ENDIF
        IF VERIFI="N"
             STORE "1" TO FLAG5
             RETURN
        ENDIF
ENDDO

* Si hay una entrada en los ficheros de clientes, se trasfiere la información
* sobre al reserva a las variable que se usan a continuación.

STORE BDATE TO MBDATE               * Fecha de la reserva de habitación
STORE LBOOK TO MLBOOK               * Longitud de la reserva
STORE ROOMNO TO MROOMNO             * Número de habitación
STORE TRIM(ROOMNO)+"/"+RTYPE TO MROOM
                                    * Clave índice formada con el número y el
                                    * tipo de la habitación

* Prepara un bloque para la pantalla de las facturas, usando direccionamiento
* de pantalla con las variables LINE y COUNT. También se incializan las
* variables COST

STORE 5 TO LINE                     * LINE es la variable de número de línea para
                                    * direccionar las líneas en la pantalla
STORE 5 TO COL                      * COL es la variable de número de columna
                                    * para direccionar las diferentes columnas
                                    * dentro de una línea de pantalla.
STORE 00000 TO TOTAL
STORE 00000 TO COST
STORE 00000 TO FTOTAL
STORE 00000 TO IVA
STORE 00000 TO RSCOST
STORE 1 TO COUNT
STORE " " TO CONT


* CARGOS TOTALES POR SERVICIO DE HABITACIONES.
* En este punto se busca en la base de datos de servicio de habitaciones, para
* establecer el valor de estos cargos.


SELECT SECONDARY
USE B:RSERVICE INDEX B:IDXSERV
FIND &MROOMNO
```

```
*  El comando FIND localiza la primera entrada de ese número de
*  habitación tomado de la base de datos de clientes y almacenado
*  en MROOMNO. Cuando se encuentra este registro, el programa se
*  mueve por la base de datos totalizando los cargos en los que
*  coincida el número de habitación.


DO WHILE #<>0 .AND. &MROOMNO=ROOMNO
    STORE COST+RSCOST TO RSCOST
    SKIP
ENDDO


*  Suma los cargos almacenados en cada registro de la habitación
*  seleccionada, hasta que se han sumado todos los cargos o hasta
*  que se llega al fin de fichero [#=0].

*  PREPARA REGISTROS TEMPORALES PARA GENERAR LA FACTURA
*  El registro usado para generar la factura se prepara en una base
*  de datos llamada FACTURAS.  Una vez que se ha generado e impreso
*  la factura, se borra el registro del fichero de FACTURAS, ya que
* no se va a necesitar más.


SELECT SECONDARY
USE B:FACTURAS
APPEND BLANK
ERASE
DO BILLSCR
```

*** Fichero BILLSCR.ZPR ***

```
  NOMBRE Sr J. Martínez                 HABITACION 234
                                        DIAS       10

+---------+----------------------------------------------+---------+---------+
| CARGO1  | 10 días a 2520 por día                       |  25200  |         |
| CARGO2  | Servicio de habitación                       |   4560  |         |
| CARGO3  | Q (no se imprime)                            |         |         |
| CARGO4  |                                              |         |         |
| CARGO5  |                                              |         |         |
| CARGO6  |                                              |         |         |
| CARGO7  |                                              |         |         |
| CARGO8  |                                              |         |         |
| CARGO9  |                                              |         |         |
| CARGO10 |                                              |         |         |
+---------+----------------------------------------------+---------+---------+
                                                         |  29760  |   3355  |
                                                         +---------+---------+
                                                                   |  31315  |
                                                                   +---------+

* Esta es la pantalla que le aparece al operados en la que se introducen los
* cargos
```

```
DO WHILE COUNT<10
    IF COUNT<10
         STORE STR(COUNT,1) TO COUNT1
    ELSE
         STORE STR(COUNT,2) TO COUNT1
    ENDIF
STORE "CARGO"+COUNT1 TO ITEM
STORE "COST"+COUNT1 TO COST
@ LINE,COL+8 GET &ITEM
READ


*  El bucle IF se usa para generar el número de cargo correcto
*  para usarlo con el registro de la factura.

    IF &ITEM="Q"
         @ 18,10 SAY "Factura completa    Pulse una tecla"
         @ 19,10 SAY "Para continuar      Pulse <RETURN> "
         @ 20,10 SAY "***********************************"
         @ 20,40 GET CONT
         READ
    ENDIF

*  Este bucle IF permite al usuario salir de la pantalla de facturas
*  si se ha introducido una "Q" en las descripción del cargo, en caso
*  contrario se siguen introduciendo datos. La posición de este bucle
*  en el programa permite al usuario salir en cualquier fase del
*  proceso de entrada de datos. La alternativae sería forzar al usuario
*  a continuar tecleando RETURN hasta que se llegara al final del
*  formulario.

    IF CONT=" "
         @ LINE,COL+55 GET &COST
         READ
         STORE &COST+TOTAL TO TOTAL
         STORE (TOTAL*12)/100 TO IVA
         STORE TOTAL+IVA TO FTOTAL
         STORE COUNT+1 TO COUNT
         STORE LINE+1 TO LINE
         @ 16,60 SAY TOTAL
         @ 16,71 SAY VAT
         @ 18,55 SAY FTOTAL
    ELSE

*  Esta opción ELSE se usa para permitir introducir los cargos por
*  servicio de habitaciones en la factura completa, antes de imprimirla.

         STORE 12 TO COUNT
         REPLACE &ITEM WITH "Servicio habitaciones"
         @ LINE,COL+8 SAY &ITEM
         @ LINE,COL+55 SAY RSCOST
         STORE &RSCOST+TOTAL TO TOTAL
         STORE (TOTAL*12)/100 TO IVA
         STORE TOTAL+IVA TO FTOTAL
         STORE COUNT+1 TO COUNT
         STORE LINE+1 TO LINE
         @ 16,60 SAY TOTAL
         @ 16,71 SAY VAT
```

```
        @ 18,55 SAY FTOTAL
        WAIT
    ENDIF
ENDDO                              * Fin del bucle DO WHILE original
ERASE
STORE " " TO BILL
@ 10,10 SAY "Pulse <RETURN> para imprimir la factura"
@ 11,10 SAY "***************************************"
@ 10,50 GET BILL
READ
SET PRINT ON
DO BILLPRT
SET PRINT OFF
SELECT PRIMARY
REPLACE CLEAR WITH "S"

*  CLEAR es un campo en la base de datos de clientes que se usa para
*  la fase de mantenimiento de ficheros. Permite borrar de la base de
*  datos maestra de clientes, las entradas cuyas facturas han sido
*  impresas. En esta fase es posible archivar esta información en un disco.

USE
SELECT SECONDARY              * B:FACTURAS
DELETE
PACK
RELEASE ALL

* Esta sección se usa para limpiar la base de datos de facturas evitando
* que se haga demasiado grande. En efecto, si usamos esta base de datos de
* esta forma, nunca será mayor de un registro.

USE
RETURN

* El comando RETURN trasfiere la ejecución del programa de vuelta al menú.
```

Este programa combina varias facilidades que lo hacen un sistema útil para aplicaciones que requieren el uso de bases de datos múltiples. En el ejemplo se genera una factura como un registro en una base de datos, y luego se imprime. Una vez impresa, el registro se libera. Las pantallas que hemos usado aquí ilustran el uso de los ficheros de comandos para crear formatos de pantalla. Estos se usan después para presentar o leer información usando la técnica del direccionamiento relativo por medio de las variables LINE y COL. Tenemos un ejemplo en BILLSCR, donde se crea un formato de factura en la pantalla y el operador introduce los detalles de la misma.

También se ilustra el uso de bases de datos primaria y secundaria para imprimir la factura, siendo el fichero primario el de CLIENTES y el secundario el de FACTURAS. En otra fase se usa como secundaria la base de datos de servicio de habitaciones. Observe la posibilidad de cambiar el fichero secundario cuando se requiere; esto se aplica también al fichero primario.

El sistema de servicio de habitaciones forma parte integral de la preparación de facturas. Se usa este sistema para introducir los cargos contra el número de habitación del cliente, cuando se ha usado el

servicio de habitaciones.

***** RSERVICE.CMD *****

Este programa se usa para introducir detalles de los cargos a las habitaciones y colocarlos por número de habitación. Una vez colocados de esta forma, el sistema de preparación de facturas puede leer estos cargos usando el número de habitación que se encuentra en la base de datos de CLIENTES.

```
ERASE
SET TALK OFF
USE B:RSERVICE INDEX B:IDXSERV
STORE " " TO CONT
STORE 4 TO LINE
STORE 2 TO POS

* Observe el uso de variables de memoria para permitir un direccionamiento
* de la pantalla más flexible.

STORE "         " TO MDATE
@ 1,2 SAY "FECHA "
@ 2,15 SAY "HAB."
@ 2,20 SAY "COD."
@ 2,32 SAY "CARGO"
@ 2,54 SAY "COSTO"
@ 3,14 SAY "--------------------------------------------"
@ 1,8 GET MDATE PICTURE "99/99/99"
@ 7,14 SAY "CODIGOS PARA CARGOS DE SERVICIO DE HABITACIONES"
@ 8,14 SAY "***********************************************"
@ 10,14 SAY "FUENTE              CODIGO    FUENTE          CODIGO"
@ 11,14 SAY "---------------------------------------------------"
@ 12,14 SAY " SALON                1       BAR                2"
@ 13,14 SAY " CENA                 3       PAPELES            4"
READ

*  La pantalla que aparece es esta:

  FECHA   /  /
              HAB. COD.        CARGO                 COSTO
              --------------------------------------------
              :    :: : :                         :  :    :


              CODIGOS PARA CARGOS DE SERVICIO DE HABITACIONES
              ***********************************************

              FUENTE              CODIGO    FUENTE          CODIGO
              ---------------------------------------------------
               SALON                1       BAR                2
               CENA                 3       PAPELES            4


DO WHILE CONT=" " .AND. LINE<=19
    APPEND BLANK
```

```
    @ LINE,POS SAY MDATE
    @ LINE,POS+13 SAY ROOMNO
    @ LINE,POS+19 SAY CODE
    @ LINE,POS+24 SAY ITEM
    @ LINE,POS+53 SAY COST
    READ
    REPLACE DATE WITH MDATE , RCLEAR WITH "N"

* El campo RCLEAR se usa para marcar un registro que no ha sido limpiado
* por la rutina de preparación de facturas.

    @ 21,5 SAY "<RETURN> para siguiente cargo, otra tecla para terminar"
    @ 22,5 SAY "*****************************************************"
    @ 22,65 GET CONT
    READ
ENDDO
RELEASE ALL              * Limpia todas las variables en memoria
ERASE
RETURN                   * Retorna al programa que lo llamó

* Esta sección de código prepara una pantalla que se usa para la entrada
* de datos. En este ejemplo se introducen los datos de uno en uno, y
* después se forma un registro en la base de datos de habitaciones.
* Usando esta técnica, cada cargo por servicio de habitación se coloca
* con un código y un número de habitación. La ventaja aparece clara cuando
* se considera la opción de informes. Por ejemplo, en la preparación de
* facturas, se encuentra el principio de las entradas de una habitación
* en particular usando el índice. Estos registros se examinan para obtener
* los totales. La alternativa sería un sumario de cargos por servicio de
* habitación por el número del código de servicio.
```

Este fichero muestra el uso de entradas en una sola línea para los cargos que pueden ocurrir varias veces y se colocan con el mismo código, en este caso el número de habitación. Cada cargo por servicio de habitación se introduce contra un número de habitación, permitiendo totalizar todos los cargos cuando se prepara la factura. El uso de un campo extra en la base de datos, para indicar si se ha hecho el cargo o no, simplifica el mantenimiento de la base de datos, ya que todos los registros que tengan puesto el campo al valor cargado se pueden borrar con un simple comando como este:

```
. DELETE ALL FOR RCLEAR="s"
```

Después de usar este comando, la base de datos debe ser empaquetada con el comando PACK, para eliminar los registros borrados, recordando poner en uso el fichero índice correcto, para que se actualice al mismo tiempo.

***** TOTSERV.CMD *****

Este programa permite examinar la base de datos de cargos por servicio de habitaciones, por el número de habitación, o por una lista de costo de servicios.

```
ERASE
SET TALK OFF
STORE 00000 TO COST1               * Prepara las variables
STORE "    " TO MROOMNO            * Número de habitación
STORE " " TO CONT
STORE " " TO ELIGE

DO WHILE CONT=" "
    @ 5,10 SAY "Introduzca número de habitación"
    @ 6,10 SAY "*******************************
    @ 5,45 GET MROOMNO
    READ
    @ 10,10 SAY " Introduzca 1 para totalizar servicio de habitación"
    @ 12,10 SAY " Introduzca 2 para un listado detallado del servicio"
    @ 13,10 SAY " ***************************************************"
    @ 12,66 GET ELIGE
    READ
    USE B:RSERVICE INDEX B:IDXSERV
    FIND &MROOMNO
    IF ELIGE="1"
         DO WHILE #<>0 .AND. MROOMNO=ROOMNO .AND. .NOT. EOF
              STORE COST+COST1 TO COST1
              SKIP
         ENDDO
         @ 15,10 SAY " Coste total de servicio de habitaciones="
         @ 15,52 SAY COST1
         ? " "
         ACCEPT "Pulse <return> para continuar" TO WAIT
    ELSE
         ERASE
         STORE 5 TO LINE
         STORE 1 TO POS
         @ LINE-4,POS SAY "Resumen de cargos por servicio de habitación"
         @ LINE-4,POS+47 SAY MROOMNO
@ LINE-2,POS SAY " FECHA     CODIGO          CARGO                        COSTO"
@ LINE-1,POS SAY "-----------------------------------------------------------"
         DO WHILE #<>0 .AND. &MROOMNO=ROOMNO .AND. .NOT. EOF
              @ LINE,POS SAY DATE
              @ LINE,POS+14 SAY CODE
              @ LINE,POS+20 SAY ITEM
              @ LINE,POS+52 SAY COST
              STORE LINE+1 TO LINE
              STORE COST+COST1 TO COST1         * Totales
              SKIP                              * Mover 1 adelante
         ENDDO
         @ LINE+1,POS+1 SAY "Total por servicio de habitación a la fecha = "
         @ LINE+1,POS+48 SAY COST1
         @ LINE+2,POS SAY " "
         ACCEPT " <RETURN> para continuar " TO WAIT
    ENDIF
ERASE
@ 10,10 SAY "Menú principal  <una tecla>"
@ 12,10 SAY "Continuar          <RETURN>"
@ 13,10 SAY "***************************"
@ 12,40 GET CONT
READ
ENDDO
RELEASE ALL EXCEPT COST1
```

```
ERASE
RETURN
```

El programa está dividido en dos secciones. Primero se obtiene el costo total por servicio de habitaciones mediante un simple bucle que añade al total el campo de costo de cada registro. Una posible alternativa sería el comando SUM. La segunda sección del programa totaliza los cargos por servicio de habitación de un número de habitación mediante la búsqueda de las entradas de esta habitación en la base de datos. A medida que se lee cada entrada, se imprimen los detalles y se actualizan los totales. Las condiciones lógicas del bucle de costo total nos aseguran que el error por fin de fichero no nos interferirá en el programa, mientras que en el segundo bucle los controles lógicos de error de fin de fichero nos aseguran que solamente se leen los registros de la habitación correcta. Ya que se está usando un fichero de índice, el comando FIND localizará la primera entrada del número de habitación. Después viene el SKIP, que nos sitúa en la siguiente entrada, totalizando los cargos, tantas veces como el campo de número de habitación contenga el número correcto.

Administración de Academia

El ejemplo que vamos a ver ahora es la parte de un sistema de administración por bases de datos. Hay varias opciones disponibles; una de las más simples es la de información de un curso. La pantalla ADMIN permite seleccionar el programa de información de curso:

***** ADMIN.CMD *****

```
ERASE
SET TALK OFF
STORE " " TO ELIGE
ERASE
@  3, 0 SAY "+----------------------------------------------------------+"
@  4, 0 SAY "|                                                          |"
@  5, 0 SAY "|                                                          |"
@  6, 0 SAY "|                Base de Datos de Administración           |"
@  7, 0 SAY "|                                                          |"
@  8, 0 SAY "+----------------------------------------------------------+"
@  9, 0 SAY "|                                                          |"
@ 10, 0 SAY "|  Están disponibles las siguientes opciones :             |"
@ 11, 0 SAY "|                                                          |"
@ 12, 0 SAY "|   1) Registro de estudiantes        6) Intro. marcas curso |"
@ 13, 0 SAY "|                                                          |"
@ 14, 0 SAY "|   2) Información de curso           7) Menú de Informes   |"
@ 15, 0 SAY "|                                                          |"
@ 16, 0 SAY "|   3) Entrada Libro de Orden         8) Salir              |"
@ 17, 0 SAY "|                                                          |"
@ 18, 0 SAY "|   4) Catálogo cintas vídeo                               |"
@ 19, 0 SAY "|                                                          |"
@ 20, 0 SAY "|   5) Catálogo Biblioteca                                 |"
@ 21, 0 SAY "|                                                          |"
@ 22, 0 SAY "|  Introduzca el número correspondiente a la opción deseada |"
@ 23, 0 SAY "|                                                          |"
@ 24, 0 SAY "+----------------------------------------------------------+"
```

```
WAIT TO ELIGE
IF ELIGE="8"
   SET TALK ON
   CANCEL
   ELSE
       IF ELIGE="1"
       DO ADMIN1
       ELSE
           IF ELIGE="2"
           DO COURSE
           ELSE
               IF ELIGE="3"
               DO NEWORDER
               ELSE
                   IF ELIGE="4"
                   DO VIDEO
                   ELSE
                       IF ELIGE="5"
                       DO LIBRARY
                       ELSE
                           IF ELIGE="6"
                           DO MARKS
                           ELSE
                               IF ELIGE="7"
                               DO REPORT
                               ENDIF
                           ENDIF
                       ENDIF
                   ENDIF
               ENDIF
           ENDIF
       ENDIF
   ENDIF
RETURN
```

Aquí se ilustra el procedimiento de selección IF; uno de sus inconvenientes es la dificultad de añadir nuevas opciones al menú. Es mucho más sencillo de hacer adiciones con la sentencia CASE en lugar de la estructura IF-ENDIF.

***** COURSE.CMD *****

El curso seleccionado se localiza mediante un campo de índice basado en un número de referencia del curso introducido en la variable CODE.

```
ERASE
SET TALK OFF
USE STUDENTS INDEX REFCODE       * Selecciona base de datos e índice
STORE "         " TO CODE        * Pone código como variable
STORE " " TO PRINT               * PRINT como variable
ERASE

@ 5, 0 SAY "+------------------------------------------------------------+"
@ 6, 0 SAY "|                                                            |"
@ 7, 0 SAY "|   PARA OBTENER DETALLES INTRODUZCA CODIGO REFERENCIA CURSO  |"
@ 8, 0 SAY "|                                                            |"
```

```
@ 9, 0 SAY "|                                                              |"
@ 10, 0 SAY "|                NUMERO DE REFERENCIA DE CURSO                 |"
@ 10,25 GET CODE
@ 10,64 SAY "|"
@ 11, 0 SAY "|                                                              |"
@ 12, 0 SAY "|                                                              |"
@ 13, 0 SAY "+--------------------------------------------------------------+"
READ                              * Leer datos del GET
STORE CODE TO REFNUM
FIND &CODE
@ 17,15 SAY "Para imprimir pulse          S"
@ 19,15 SAY "Para pantalla solo, pulse    N"
WAIT TO PRINT

IF PRINT="N"
        DO CSCREEN3               * Mostrar información del
                                  * curso en la pantalla
        DO ADMIN                  * Volver al menú principal
ELSE
        DO CSCREEN4               * Mostrar información en pantalla
                                  * y seleccionar una impresión
                                  * de esa información

        SET PRINT ON              * Mandar la salida a la impresora
        DO CSCREEN2
        SET PRINT OFF             * Volver a mandar a pantalla
        DO ADMIN
        READ
        RETURN                    * Volver al menú principal
ENDIF
```

***** CSCREEN4.CMD *****

Pantalla llamada desde el fichero de comando COURSE:

```
ERASE
@  1, 0 SAY "+--------------------------------------------------------------+"
@  2, 0 SAY "|                                                              |"
@  3, 0 SAY "|        TITULO DEL CURSO"
@  3,35 SAY TITLE
@  3,64 SAY "|"
@  4, 0 SAY "|                                                              |"
@  5, 0 SAY "|        NUMERO DE REFERENCIA"
@  5,35 SAY REFNUM
@  5,64 SAY "|"
@  6, 0 SAY "|                                                              |"
@  7, 0 SAY "|        FECHA DEL CURSO"
@  7,35 SAY DATE
@  7,64 SAY "|"
@  8, 0 SAY "|                                                              |"
@  9, 0 SAY "|        LONGITUD DEL CURSO"
@  9,35 SAY LENGHT
@  9,64 SAY "|"
@ 10, 0 SAY "|                                                              |"
@ 11, 0 SAY "|        COSTO DEL CURSO"
@ 11,35 SAY COST
@ 11,64 SAY "|"
```

```
@ 12, 0 SAY "|                                                              |"
@ 13, 0 SAY "|         NUMERO DE ESTUDIANTES"
@ 13,35 SAY NUMBER
@ 13,64 SAY "|"
@ 14, 0 SAY "|                                                              |"
@ 15, 0 SAY "+--------------------------------------------------------------+"
@ 17, 0 SAY "Para obtener una copia impresa, pulse una tecla"
WAIT
RETURN
```

Libro de Pedidos de Almacén

Esta es una aplicación muy sencilla y usa pocas facilidades del dBaseII. Obtiene la información que está presente en una base de datos, y al saca a la pantalla o a la impresora.

El nuevo sistema de pedidos se llama ORDBOOK.CMD.

***** ORDBOOK.CMD *****

```
ERASE
SET TALK OFF
ERASE

@  2, 0 SAY "+--------------------------------------------------------------+"
@  3, 0 SAY "|                                                              |"
@  4, 0 SAY "|                                                              |"
@  5, 0 SAY "|                                                              |"
@  6, 0 SAY "|                       LIBRO DE PEDIDOS                       |"
@  7, 0 SAY "|                                                              |"
@  8, 0 SAY "+--------------------------------------------------------------+"
@  9, 0 SAY "|                                                              |"
@  0, 0 SAY "|  ESTAN DISPONIBLES LAS SIGUIENTES FUNCIONES                  |"
@ 11, 0 SAY "|                                                              |"
@ 12, 0 SAY "|        1) INTRODUCIR UN NUEVO PEDIDO                         |"
@ 13, 0 SAY "|                                                              |"
@ 14, 0 SAY "|        2) BUSCAR UN PEDIDO ANTIGUO                           |"
@ 15, 0 SAY "|                                                              |"
@ 16, 0 SAY "|        3) SALIR                                              |"
@ 17, 0 SAY "|                                                              |"
@ 18, 0 SAY "|  INTRODUZCA EL NUMERO CORRESPONDIENTE A LA OPCION DESEADA    |"
@ 19, 0 SAY "|                                                              |"
@ 20, 0 SAY "+--------------------------------------------------------------+"
RETURN
```

Esta pantalla permite al operador seleccionar la acción requerida y el programa NEWORDER realizará la selección.

***** NEWORDER.CMD *****

```
ERASE
SET TALK OFF
USE ORDER1 INDEX ORDIDX
STORE " " TO ELIGE3
STORE "S" TO MORE
```

```
ERASE
DO ORDBOOK
WAIT ELIGE3
STORE " " TO PRINT
IF ELIGE3 = "1"
          DO WHILE ELIGE3 = "1" .AND. MORE = "S"
          APPEND BLANK
          DO FORDER
          IF POST = "P"
               REPLACE TYPE WITH "P"
               ELSE
               REPLACE TYPE WITH "C"
          ENDIF
STORE AMOUNT1+AMOUNT2 TO AM8
STORE AMOUNT3+AMOUNT4+AMOUNT5 TO AM9
STORE AMOUNT6+AMOUNT7 TO AM10
STORE AM8+AM9+AM10 TO MTOTAL
REPLACE IVA WITH (MTOTAL*0.12)
REPLACE TOTAL WITH (TOTAL+IVA)
STORE "N" TO PRINT
ACCEPT "¿Quiere copia impresa de este pedido?  S\N " TO PRINT


IF PRINT = "S"
          DO ORDER
          @ 27,10 SAY "Pulse una tecla para empezar la impresión"
          WAIT
          SET PRINT ON
          DO PRODER
          SET PRINT OFF
ELSE
          DO ORDER
ENDIF

ACCEPT "¿Más pedidos nuevos? S\N " TO MORE
ENDDO
RELEASE ALL
DO NEWORDER
ELSE
IF ELIGE3 = "3"
    DO WHILE ELIGE3 = "2" .AND. MORE = "S"
          STORE "           " TO MORDNUM
          ACCEPT "Introduzca el número de pedido  " TO MORDNUM
          FIND &MORDNUM
          ERASE
          ACCEPT "¿Quiere una copia impresa de este pedido? S\N " TO PRINT
          IF PRINT = "S"
               DO ORDER
               @ 27,10 SAY "Pulse una tecla para empezar la impresión"
               WAIT
               SET PRINT ON
               DO PRODER
               SET PRINT OFF
          ELSE
               DO ORDER
          ENDIF
@ 57,23 SAY "¿Más peticiones? S\N "
WAIT TO MORE
```

```
ENDDO
RELEASE ALL
DO NEWORDER
ELSE
IF ELIGE3 = "3"
ENDIF
ENDIF
ENDIF
DO ADMIN
RETURN
```

Si se va a introducir un nuevo pedido, aparecerá la pantalla de pedidos y se introducirán los datos individuales. Si se va a buscar un pedido, se introducirá el número de pedido y se ejecutará un FIND por la base de datos. Este pedido requerido aparecerá en la pantalla y se podrá obtener una copia impresa.

***** PORDER.CMD *****

```
SET FORMAY TO PRINT                         * Mandar salida a impresora
SET MARGIN TO 0

ERASE
@  1, 0 SAY "+------------------------------------------------------------------------------+"
@  2, 0 SAY "|"
@  2,10 SAY APELLIDO
@  2,53 SAY "Nø PEDIDO"
@  2,62 SAY ORDNUM
@  2,79 SAY "|"
@  3, 0 SAY "|                                                                              |"
@  4, 0 SAY "|"
@  4,10 SAY DIR1
@  4,53 SAY "FECHA"
@  4,65 SAY DATE
@  4,79 SAY "|"
@  5, 0 SAY "|                                                                              |"
@  6, 0 SAY "|"
@  6,10 SAY DIR2
@  6,79 SAY "|"
@  7, 0 SAY "|                                                                              |"
@  8, 0 SAY "|"
@  8,10 SAY DIR3
@  8,79 SAY "|"
@  9, 0 SAY "+------------------------------------------------------------------+-----------+"
@ 10, 0 SAY "|    POR FAVOR SUMINISTRE                                          |   TOTAL   |"
@ 11, 0 SAY "+------------------------------------------------------------------+-----------+"
@ 12, 0 SAY "| 1"
@ 12, 5 SAY DES1
@ 12,67 SAY "|"
@ 12,68 SAY AMOUNT1
@ 12,79 SAY "|"
@ 13, 0 SAY "| 2"
@ 13, 5 SAY DES2
@ 13,67 SAY "|"
@ 13,68 SAY AMOUNT2
@ 13,79 SAY "|"
@ 14, 0 SAY "| 3"
```

```
@ 14, 5 SAY DES3
@ 14,67 SAY "|"
@ 14,68 SAY AMOUNT3
@ 14,79 SAY "|"
@ 15, 0 SAY "| 4"
@ 15, 5 SAY DES4
@ 15,67 SAY "|"
@ 15,68 SAY AMOUNT4
@ 15,79 SAY "|"
@ 16, 0 SAY "| 5"
@ 16, 5 SAY DES5
@ 16,67 SAY "|"
@ 16,68 SAY AMOUNT5
@ 16,79 SAY "|"
@ 17, 0 SAY "| 6"
@ 17, 5 SAY DES6
@ 17,67 SAY "|"
@ 17,68 SAY AMOUNT6
@ 17,79 SAY "|"
@ 18, 0 SAY "| 7"
@ 18, 5 SAY DES7
@ 18,67 SAY "|"
@ 18,68 SAY AMOUNT7
@ 18,79 SAY "|"
@ 19, 0 SAY "|                                ESTE PEDIDO HA ORIGINADO UN IVA(12%) |"
@ 19,68 SAY IVA
@ 19,79 SAY "|"
@ 20, 0 SAY "+------------------------------------------------------------------+-----------+"
@ 21, 0 SAY "|    CONFIRMADA O POR CORREO"
@ 21,30 SAY TYPE
@ 21,55 SAY "TOTAL        |"
@ 21,68 SAY TOTAL
@ 21,79 SAY "|"
@ 22, 0 SAY "+------------------------------------------------------------------+-----------+"
SET FORMAT TO SCREEN                          * Volver a la pantalla
RETURN
```

El sistema de mantenimiento de registros de pedidos se puede programar de acuerdo a los requerimientos de una escuela en particular, academia o institución.

Seguro de Coche

El sistema de ficheros usados para la póliza de seguro para coche consiste en los siguientes ficheros:

Bases datos	Indices	Pantallas	Formatos
CUSTIN	IDXNAME	SCRCAR1	SCRCAR1E
CARBASE	CARNUMB	SCRCAR2	SCRCAR2E
		SCRCAR1D	SCRCARP
		SCRCAR2D	

Los ficheros de la base de datos contienen información del cliente y el número de la póliza y están indexados por el nombre del cliente y por el número de póliza respectivamente. Los ficheros de pantalla

están divididos en entrada de datos (SCRCAR1 y 2) y en presentación de datos (SCRCAR1D y 2D). Los ficheros usados para formatear los registros editados (SCRCAR1E y 2E) son ficheros de formato y el fichero SCRCARP es el fichero usado para imprimir el formulario completo.

***** Fichero SCREEN1.ZPR *****

```
                                      SEGUROS

------------------------------------------------------------------------------
     INTRODUZCA NUEVA POLIZA             MODIFICAR o CANCELAR POLIZA
     -----------------------             ---------------------------
     A1) COCHE                                   B1) MODIFICAR
     A2) PERSONAL                                B2) CANCELAR
     A3) VIVIENDA
     A4) GENERAL

     BUSCAR POLIZA                            SALIR DEL SISTEMA
     -------------                            -----------------
     C1) POR NOMBRE DE CLIENTE                       D1) SALIR
     C2) POR NUMERO DE POLIZA
------------------------------------------------------------------------------
     INTRODUZCA LA ACCION REQUERIDA #ACTION
```

El formato del menú que hemos usado aquí es similar al ilustrado en el sistema de reservas de hotel. La ventaja de este sistema reside en la facilidad de seleccionar la opción requerida desde una pantalla en lugar de usar una sucesión de pantallas para ejecutar el programa de aplicación.

***** SCREEN1.CMD ***   Fichero de comandos del Menú**

```
ERASE
SET TALK OFF
STORE " " TO ACTION
STORE "N" TO ELIGE
@ 1,38 SAY "SEGUROS"
@ 2, 1 SAY "-------------------------------------------------------"
@ 2,56 SAY "-----------------------"
@ 3, 6 SAY "INTRODUZCA NUEVA POLIZA            MODIFICAR o CANCELAR"
@ 3,62 SAY "POLIZA"
@ 4, 6 SAY "-----------------------            --------------------"
@ 4,61 SAY "-------"
@ 5, 6 SAY "A1) COCHE                                   B1) MODIFIC"
@ 5,61 SAY "AR"
@ 6, 6 SAY "A2) PERSONAL                                B2) CANCELA"
@ 6,61 SAY "R"
@ 7, 6 SAY "A3) VIVIENDA"
@ 8, 6 SAY "A4) GENERAL"
@ 10, 6 SAY "BUSCAR POLIZA                            SALIR DEL SISTE"
@ 10,61 SAY "MA"
@ 11, 6 SAY "-------------                            ---------------"
@ 11,61 SAY "--"
@ 12, 6 SAY "C1) POR NOMBRE DE CLIENTE                       D1) SALIR"
@ 13, 6 SAY "C2) POR NUMERO DE POLIZA"
```

```
@ 14, 1 SAY "----------------------------------------------------"
@ 14,56 SAY "-----------------------"
@ 15, 6 SAY "INTRODUZCA LA ACCION REQUERIDA"
@ 15,38 GET ACTION PICTURE "A9"              * El uso de la cláusula PIC
READ                                         * asegura el orden letra,número
CLEAR GETS                                   * Buena práctica limpiar los GET


* Sistema de detección de errores


DO WHILE CHECK="N"
STORE $(ACTION,1,1) TO SELECT                * Parte alfabética de ACTION
STORE $(ACTION,2,1) TO TYPE                  * Parte numérica de ACTION


IF .NOT. (SELECT="A" .OR. SELECT="B" .OR. SELECT="C" .OR. SELECT="D")


* Lógica que permite comprobar la letra del código


    @ 17,5 SAY "Acción inválida   REPITA"
    @ 17,50 GET ACTION PIC "A9"
    READ
    CLEAR GETS
ELSE
    STORE "S" TO CHECK
ENDIF
ENDDO

*  Una alternativa sería incluir la comprobación de errores
*  en una sentencia CASE y usar la cláusula OTHERWISE para
*  ejecutar la recuperación

DO CASE
    CASE SELECT="A"
         DO CUSTIN
         IF TYPE="1"
              DO CAR
              ELSE
                   IF TYPE="2"
                        DO PERSON
                        ELSE
                             IF TYPE="3"
                                  DO HOUSE
                                  ELSE
                                       IF TYPE="4"
                                            DO GEN
                                       ENDIF
                             ENDIF
                   ENDIF
         ENDIF
    CASE SELECT="B"
         DO CHANGE
    CASE SELECT="C"
         DO SEARCH
    CASE SELECT="D"
```

```
        QUIT
        RETURN
ENDCASE
```

***** SCREEN2.ZPR *****

Esta pantalla se usa para introducir detalles del cliente y se llama automáticamente cuando se introduce una póliza nueva.

```
                              DETALLES DEL CLIENTE

--------------------------------------------------------------------------------
Sr\Sra\Srta                                   CALLE

APELLIDOS                                     DISTRITO

NOMBRE                                        CIUDAD

FECHA de NACIMIENTO                           REGION

PROFESION                                     CODIGO POSTAL

Nº TELEFONO (casa)                            (oficina)

Nº SUCURSAL                                   Nº CLIENTE

--------------------------------------------------------------------------------

ERASE
SET TALK OFF
STORE "N" TO CHECK
STORE " " TO VERIFY
DO SCREEN2
STORE "N" TO REPEAT
USE CUSTIN INDEX IDXNAME
APPEND BLANK
DO WHILE CHECK='N'
    @  4,16 GET SALUTE
    @  6,16 GET APELLID
    @  8,16 GET NOMBRE
    @ 10,24 GET DOFB
    @ 12,16 GET OCCUP
    @ 14,24 GET TELEH
    @ 16,24 GET BRNUM
    @  4,59 GET DIR1
    @  6,59 GET DIR2
    @  8,59 GET DIR3
    @ 10,59 GET DIR4
    @ 12,62 GET PCODE
    @ 14,62 GET TELEB
    @ 16,62 GET CNUMB
    READ
    CLEAR GETS
    @ 19,10 SAY "
    @ 19,10 SAY "¿Son correctos los datos? <S> o <N>"
    @ 19,50 GET VERIFY
    READ
```

```
    CLEAR GETS
         IF VERIFY="N"
              @ 19,10 SAY "                                        "
              @ 19,10 SAY "Mueva el cursor al campo y corríjalo"
         ELSE
              STORE "S" TO CHECK
         ENDIF
ENDDO
```

```
* Este programa usa una pantalla preparada por SCREEN2 y lee los datos
* de la parte izquierda de la pantalla desde arriba hacia abajo, y
* después los de la parte derecha desde arriba hacia abajo también,
* dando un orden más natural de entrada.
```

*** CAR.CMD ***

Este programa permite introducir los detalles de una póliza mediante dos pantallas. SCRCAR1 y 2. Estas pantallas contienen un mensaje que permite introducir datos que se pueden modificar antes de salvar el registro. También permite un opción de imprimir, usando el fichero SCRCARP, que nos da un informe del formulario completo combinando SCRCAR1 y 2.

```
ERASE
USE B:CARBASE INDEX B:CARNUMB              * Base de datos de pólizas de
                                           * coches e índice
STORE "N' TO CHECK
DO WHILE CHECK="N"
    APPEND BLANK
    DO SCRCAR1                             * Primera pantalla de entrada de datos
         IF CHECK="N"
              @ 20,10 SAY "POSICIONE EL CURSOR SOBRE EL CAMPO Y CAMBIELO"
         ENDIF                             * Si CHECK="N", permite los
                                           * cambios
ENDDO
STORE "N" TO CHECK
DO WHILE CHECK="N"
    APPEND BLANK
    DO SCRCAR2                             * Segunda pantalla de entrada de datos
         IF CHECK="N"
              @ 20,10 SAY "POSICIONE EL CURSOR SOBRE EL CAMPO Y CAMBIELO"
         ENDIF                             * Si CHECK="N", permite los
                                           * cambios
ENDDO
ERASE
TEXT                                       * Use un TEXT-ENDTEXT en lugar
                                           * de @y SAY

   PARA OBTENER UNA COPIA IMPRESA        P

   NO HACE FALTA COPIA, VOLVER AL MENU  M

ENDTEXT
DO WHILE .NOT. (ACTION="P" .OR. ACTION="M")
    @ 15,10 SAY "INTRODUCIR ELECCION   "
    @ 15,32 GET ACTION PICTURE "A"
```

```
    READ
ENDDO

* El bucle DO WHILE asegura que el paso a la siguiente parte del programa
* ocurra solamente cuando se seleccione una opción correcta.

IF ACTION="P"
    SET PRINT ON
    SET FORMAT TO PRINT
    DO SCRCARP
    SET FORMAT TO SCREEN
    SET PRINT OFF
ELSE
    DO SCREEN1
ENDIF
```

SCRCARP.CMD es el fichero usado para imprimir el formulario entero y solamente consiste en cláusulas SAY y USING. La impresora saltará primero una página en blanco; esto se puede evitar usando el comando SET EJECT OFF.

***** SEARCH.CMD *****

SEARCH.CMD permite al usuario buscar una póliza específica por número de póliza. Una alternativa sería buscarla por el nombre del cliente, usando la base de datos que contiene un registro por cada número de póliza de un cliente. Una vez encontrada, los detalles de la póliza se presentan en la pantalla y se puede imprimir o copiar.

```
ERASE
SET TALK OFF
USE B:CARBASE INDEX B:CARNUMB
STORE " " TO ACTION
STORE "N" TO CHECK
STORE "R" TO OPTION
STORE "      " TO MPNUMB                    * Variable índice - el uso del
                                             * prefijo M la identifica como
                                             * una variable en memoria
TEXT
    SI SE CONOCE EL NUMERO DE POLIZA, SE PUEDE ENCONTRAR UNA

    POLIZA DETERMINADA. SI NO CONOCE EL NUMERO DE POLIZA,

    VUELVE AL MENU PRINCIPAL Y SELECCIONE BUSCAR POR NOMBRE.

ENDTEXT
DO WHILE OPTION="R"
    DO WHILE CHECK="N"
        @ 15,10 SAY " C)  PARA CONTINUAR   M)  PARA VOLVER AL MENU "
        @ 15,65 GET ACTION PICTURE "A"
        READ
            IF .NOT. (ACTION="C" .OR. ACTION="M")
                @ 17,10 SAY " INTRODUZCA C o M  "
                @ 18,10 SAY " ***************  "
            ELSE
                STORE "S" TO CHECK
            ENDIF
```

```
    ENDDO
    STORE "N" TO CHECK
    ERASE
    @ 10,10 SAY "INTRODUZCA NUMERO DE POLIZA"
    @ 10,42 GET MPNUMB PICTURE "A/9999"
    READ
    FIND &MPNUMB                    * Búsqueda basada en el valor de MPNUMB
         IF #=0
              ERASE
              @ 10,10 SAY "NO EXISTE UNA POLIZA CON ESE NUMERO"
              @ 12,10 SAY "R)   VOLVER A INTRODUCIR    Q) VOLVER AL MENU"
              @ 12,70 GET OPTION PICTURE "A"
              READ
         ELSE
              STORE " " TO OPTION
         ENDIF
ENDDO
RELEASE OPTION,CHECK                * Se elimina de memoria las variables
                                    * OPTION y CHECK.
STORE "S" TO VIEW                   * Prepara nuevas variables en memoria
STORE " " TO CONT
DO WHILE VIEW="S"
         ERASE
         DO SCRCAR1D                * Presenta página 1 y un mensaje
                                    * para la página 2
         @ 20,10 SAY " N) SIGUIENTE PAGINA    Q) PARA SALIR"
         @ 20,60 GET CONT
         READ
              IF CONT="N"           * Si no hay segunda página, volver al menú
                   DO SCRCAR2D
              ELSE
                   RETURN
              ENDIF
         @ 22,10 SAY "N) SIGUIENTE POLIZA     C) COPIAR POLIZA"
         @ 22,70 GET VIEW
         READ
              IF VIEW="N"
                   SKIP

*  Si se requiere la siguiente póliza, salte un registro hacia adelante. El
*  siguiente registro que se muestra no será necesariamente el contador de
*  registros más 1; será el siguiente registro en el orden del índice.

                   STORE "S" TO VIEW
              ELSE
                   SET EJECT OFF
                   SET PRINT ON
                   SET FORMAT TO PRINT
                   DO SCRCARP
                   SET FORMAT TO SCREEN
                   @ 10,10 SAY " LA COPIA ESTA HECHA "
                   @ 11,10 SAY " PULSE RETURN PARA EL MENU PRINCIPAL"
                   @ 11,75 GET VIEW
                   READ
              ENDIF
ENDDO
RELEASE VIEW,CONT                   * Limpiar la memoria antes de
                                    * volver al menú principal
```

***** CHANGE.CMD *****

El programa CHANGE.CMD permite al usuario alterar el contenido de una póliza por medio de una pantalla de edición formateada. La única diferencia entre este programa y el de búsqueda es la sección que llama al fichero FMT usado para editar.

```
ERASE
SET TALK OFF
USE B:CARBASE INDEX B:CARNUMB
STORE " " TO ACTION
STORE "N" TO CHECK
STORE "R" TO OPTION
STORE "      " TO MPNUMB
TEXT
    SI SE CONOCE EL NUMERO DE POLIZA, SE PUEDE ENCONTRAR UNA

    POLIZA DETERMINADA. SI NO CONOCE EL NUMERO DE POLIZA,

    VUELVE AL MENU PRINCIPAL Y SELECCIONE BUSCAR POR NOMBRE.

ENDTEXT
DO WHILE OPTION="R"
    DO WHILE CHECK="N"
         @ 15,10 SAY " C)  PARA CONTINUAR   M)  PARA VOLVER AL MENU "
         @ 15,65 GET ACTION PICTURE "A"
         READ
              IF .NOT. (ACTION="C" .OR. ACTION="M")
                   @ 17,10 SAY " INTRODUZCA C o M  "
                   @ 18,10 SAY " ****************  "
              ELSE
                   STORE "S" TO CHECK
              ENDIF
    ENDDO
    STORE "N" TO CHECK
    ERASE
    @ 10,10 SAY "INTRODUZCA NUMERO DE POLIZA"
    @ 10,42 GET MPNUMB PICTURE "A/9999"
    READ
    FIND &MPNUMB                        * Búsqueda basada en el valor de MPNUMB
         IF #=0
              ERASE
              @ 10,10 SAY "NO EXISTE UNA POLIZA CON ESE NUMERO"
              @ 12,10 SAY "R)  VOLVER A INTRODUCIR   Q) VOLVER AL MENU"
              @ 12,70 GET OPTION PICTURE "A"
              READ
         ELSE
              STORE " " TO OPTION
         ENDIF
ENDDO
RELEASE OPTION,CHECK
STORE "S" TO VIEW
STORE " " TO CONT
DO WHILE VIEW="S"
         ERASE
         SET FORMAT TO SCRCARE          * Selecciona el fichero de formato de edición
              DO WHILE CHEK="N"
                   EDIT #               * Edita el registro
```

```
            ENDDO
        SET FORMAT TO SCREEN


        @ 20,10 SAY " N) SIGUIENTE PAGINA    Q) PARA SALIR"
        @ 20,60 GET CONT
        READ
            IF CONT="N"
                SET FORMAT TO SCRCAR2E
                    DO WHILE CHEK="N"
                        EDIT #
                    ENDDO
                SET FORMAT TO SCREEN
            ELSE
                RETURN
            ENDIF
        @ 22,10 SAY "N) SIGUIENTE POLIZA     C) COPIAR POLIZA"
        @ 22,70 GET VIEW
        READ
            IF VIEW="N"
                SKIP
                STORE "S" TO VIEW
            ELSE
                SET EJECT OFF           * Elimina el salto de página
                SET PRINT ON            * Activa la impresora
                SET FORMAT TO PRINT     * Pone formato impresión
                DO SCRCARP
                SET FORMAT TO SCREEN    * Pone formato pantalla


                @ 10,10 SAY " LA COPIA ESTA HECHA "
                @ 11,10 SAY " PULSE RETURN PARA EL MENU PRINCIPAL"
                @ 11,75 GET VIEW
                READ
            ENDIF
ENDDO
RELEASE VIEW,CONT
RETURN                                  * Vuelve al programa que lo llamó
```

El programa que le hemos enseñado aquí ilustra la técnica usada para preparar una aplicación. Además de las facilidades de los programas, nos será de ayuda una rutina para actualizar el registro de cliente con los números de las pólizas. Esto permitirá buscar en la base de datos de clientes por nombre o por número de póliza, después se busca en la base de datos de pólizas para obtener los detalles. La razón principal es la situación en la que un cliente no conoce su número de póliza pero sí sabe para lo que es la póliza. Se requieren un gran números de campos para contener toda la información que necesita el formulario, y en este ejemplo se agrupan varios datos dentro de un mismo campo. La mejor solución es usar dos bases de datos cuyos registros estén conectados por número de póliza. En el siguiente ejemplo de entrada de pedidos se usan bases de datos paralelas.

Introducción y comprobación de pedidos

Los siguientes programas muestran un sistema de introducción de pe-

didos que hace uso de bases de datos paralelas y pantallas que direccionan las áreas de las bases de datos primaria y secundaria. Cada registro en la base de datos debe tener un campo de orden, identificado por un número de orden y un número de línea.

```
ERASE
SET TALK OFF
STORE " " TO CHECK
STORE "S" TO GO
DO WHILE GO="S"

** Las siguientes líneas muestran el menú

    @ 4,10 SAY "TIENE DISPONIBLES LAS SIGUIENTES OPCIONES"
    @ 6,10 SAY "1) INTRODUCIR PEDIDO NUEVO
    @ 8,10 SAY "2) BUSCAR PEDIDO ANTIGUO"
    @ 10,10 SAY "3) MODIFICAR UN PEDIDO EXISTENTE"
    @ 12,10 SAY "4) SALIR DEL SISTEMA"
    @ 14,10 SAY "INTRODUZCA EL NUMERO ELEGIDO" TO CHOICE
    @ 14,60 GET CHOICE
    READ

** Sentencia CASE para la selección del fichero de comandos requerido

    DO CASE
         CASE CHOICE="1"
              DO ORDERIN              * Introducir un pedido
         CASE CHOICE="2"
              DO ORDEROUT             * Mostrar un pedido
         CASE CHOICE="3"
              DO EDIT                 * Editar un pedido
         CASE CHOICE="4"
              STORE "N" TO GO         * Salir cambiando GO a N
    ENDCASE
    ERASE
ENDDO
```

***** ORDERIN.CMD *****

Este programa introduce los datos de un pedido en la base de datos.

```
SET FORMAT TO SCREEN
ERASE                              * Limpiar la pantalla
SET TALK OFF
SET COLON OFF                      * Evita mostrar los marcadores de campo
RESTORE FROM PARAM1

*  El comando RESTORE lee el fichero de memoria PARAM1 y restablece
*  las variables de memoria contenidas en ese fichero. Las variables
*  existentes se borran. Si se usa la opción ADDITIVE, no se borrarán.

SELECT PRIMARY                     * Base de datos primaria e índice
USE &DBASE INDEX &INDEX

*  DBASE e INDEX son variable de memoria contenidas en el fichero PARAM1
*  Usando esta técnica se pueden cambiar los nombres de las bases de datos
*  fácilmente en el fichero de parámetros, y no se necesita cambiar el
```

```
*  código del programa.

SELECT SECONDARY             * Base de datos secundaria e índice
USE &DBASE1 INDEX &INDEX1    * Similar al comando usado en la primaria
SELECT PRIMARY               * Trabajar con la base de datos primaria
APPEND BLANK                 * Añadir un registro en blanco
DO ORDERENTR                 * Preparar la pantalla de pedidos
@ 0,13 GET ORDNUM            * Leer número de pedido

* Se puede usar una cláusula PICTURE para formatear la entrada
* del número de pedido.

@ 0,41 GET DATE PICTURE "99/99/99"
@ 0,66 GET SUPP:CODE
READ

DO WHILE COUNT<15 .AND. CONT="S"
    IF COUNT<10
         STORE STR(COUNT,1,0) TO COUNT1
    ELSE
         STORE STR(COUNT,2,0) TO COUNT1
    ENDIF

* Este bucle genera el número de secuencia de la entrada

    STORE "ITEM"+COUNT1 TO ITEM
    STORE "QUANT"+COUNT1 TO QUANT
    STORE "U:COST"+COUNT1 TO U:COST
    STORE "TOTAL"+COUNT1 TO TOTAL

* Los comandos STORE crean variables de memoria con los nombres de los campos

    @ LINE,6 GET &ITEM
    @ LINE,47 GET &QUANT PICTURE "99999"
    @ LINE,55 GET &U:COST
    READ
    CLEAR GETS
    STORE (&QUANT+&U:COST) TO CTOTAL

         DO WHILE CTOTAL>999999.99
              @ 19,1 SAY "EL VALOR TOTAL EXCEDE DE 999999.99"
              @ 20,1 SAY "INTRODUZCALO EN DOS CANTIDADES PEQUEÑAS"
              @ LINE,47 GET &QUANT
              READ
              @ 19,1 SAY "                                  "
              @ 20,1 SAY "                                       "
         ENDDO

* Este bucle comprueba el tamaño del campo CTOTAL y da un error si excede
* del tamaño prefijado. Una comprobación de este tipo previene un error
* por rebasamiento numérico.

    STORE CTOTAL TO ITOTAL
    REPLACE &TOTAL WITH ITOTAL
    STORE RTOTAL+ITOTAL TO RTOTAL
    @ LINE,66 SAY &TOTAL
    STORE COUNT+1 TO COUNT
    STORE LINE+1 TO LINE
```

```
        IF COUNT=8 .AND. CONT="S"
             REPLACE ORD:NO WITH ORDNUM
             SELECT SECONDARY
             APPEND BLANK
        ENDIF

* Si el puntos de items excede de 8, se abre la base de datos secundaria
* se añade un nuevo registro para contener los puntos adicionales. El
* comando REPLACE se usa para trasferir el número de pedido, de forma
* que el registro se puede encontrar en operaciones de búsqueda posteriores.

    @ 20,1 SAY "CONTINUA INTRODUCIENDO DATOS  SI(S)  NO(N)"
    @ 20,43 GET CONT PICTURE "A"
    READ
        IF CONT="S"
             @ 20,1 SAY "                                             "
        ENDIF

** Limpia el mensaje en la línea 20

ENDDO
@ 19,66 SAY RTOTAL
SELECT PRIMARY                          * Vuelve al fichero primario
FIND &ORDNUM                            * busca el registro actual de pedido
REPLACE VALUE WITH RTOTAL               * Introduce el valor total en el pedido

@ 20,1 SAY "                                             "
@ 19,1 SAY "¿SON CORRECTOS LOS DETALLES?"
@ 19,30 GET VERIFY
READ

    IF VERIFY="S"
         USE B:ORDNUMB
         APPEND BLANK
         REPLACE ORD:NO WITH ORDNUM,AVAIL WITH "N"
         RETURN
    ELSE
         DO EDIT
    ENDIF

* La última sección usa una base de datos de número de pedido y crea
* un registro que contiene el número de pedido y un campo llamado AVAIL.
* Este campo muestra el estado del número de pedido. la base de datos
* ORDNUM se puede usar para comprobar si se ha usado un número de pedido,
* antes de escribirlo. Si se requieren cambios en el pedido, se llama al
* fichero EDIT y se edita el pedido.
```

*** ORDEROUT.CMD ***

Este fichero de comandos se usa para mostrar en pantalla el pedido registrado con un número de pedido que le da el operador.

```
ERASE                                   * Limpia la pantalla
SET TALK OFF
STORE "S" TO MORE                       * Prepara MORE
USE B:ORDERS INDEX B:ORDIDX1            * Selecciona base de datos e índice
DO WHILE MORE="S"
```

```
    ERASE
    STORE "          " TO FINDER
    @ 10,10 SAY "INTRODUZCA EL NUMERO DE PEDIDO QUE DESAE BUSCAR"
    @ 10,60 GET FINDER
    READ
    ERASE
    FIND &FINDER                        * Busca el registro
    DO ORDERIN                          * Prepara la pantalla

    @ 0,13 SAY ORD:NO
    @ 0,41 SAY DATE USING "99/99/99"
    @ 0,66 SAY SUPP:CODE

* Muestra el número de pedido, fecha de pedido y código del proveedor.

    STORE 4 TO LINE
    STORE 1 TO COUNT

* Prepara las variables de línea y columna

        DO WHILE COUNR<15
            IF COUNT<10
                STORE STR(COUNT,1,0) TO COUNT1
            ELSE
                STORE STR(COUNT,2,0) TO COUNT1
            ENDIF

* Bucle para preparar la parte del número de campo del nombre del campo

            STORE "ITEM"+COUNT1 TO COUNT1
            STORE "QUANT"+COUNT1 TO QUANT
            STORE "U:COST"+COUNT1 TO U:COST
            STORE "TOTAL"+COUNT1 TO TOTAL

                IF &ITEM<>' ' .OR. &TOTAL<>0.00
                    @ LINE,6 SAY &ITEM
                    @ LINE,47 SAY &QUANT USING "99999"
                    @ LINE,55 SAY &U:COST
                    @ LINE,66 SAY &TOTAL
                ENDIF

            STORE COUNT+1 TO COUNT
            STORE LINE+1 TO LINE

                IF COUNT=0
                    USE B:ORDERS1 INDEX B:ORDIDX2
                    FIND &FINDER
                        IF #=0
                            STORE 15 TO COUNT
                        ENDIF
                ENDIF
        ENDDO
* La última sección de este bucle DO WHILE cambia de base de datos si
* hay más de ocho puntos en el pedido. Si #=0 es que se ha llegado al
* fin de fichero por defecto.

USE B:ORDERS INDEX B:ORDICX1
FIND &FINDER
```

```
* Vuelve a la base de datos original para obtener el valor total del pedido

@ 19,65 SAY VALUE
STORE " " TO MORE
@ 20,3 SAY "¿CONTINUO LA BUSQUEDA? ------<S> O <N>"
@ 20,40 GET MORE
READ
ENDDO
RETURN
```

***** EDIT.CMD *****

Este fichero se usa para editar un registro que contiene un pedido.

```
ERASE
SET TALK OFF
USE B:ORDER1 INDEX B:ORDNUMB            * Selecciona base de datos e índice
ACCEPT "NUMERO DE PEDIDO " TO ORDNUM    * Introducir el número de pedido
FIND &ORDNUM                            * Busca el pedido
SET FORMAT TO ORDEDIT                   * Prepara el formato con un fichero
                                        * para editar el registro
EDIT #                                  * Edita el registro actual

SET FORMAT TO SCREEN
RETURN
```

El programa EDIT buscará un registro en la base de datos de pedidos. Si existe otra parte del pedido en la base de datos paralela, se necesitará el mismo código adicional que se usó para escribir la información en la base de datos.

Pantalla de entrada de pedidos, tal y como aparece.

***** ORDENTR.ZPR *****

```
NUMERO PEDIDO          FECHA PEDIDO          CODIGO DEL PROVEEDOR

PUNTO                  DESCRIPCION            CANTIDAD COSTO UNID   TOTAL
-----+-----------------------------------+--------+----------+-----------+
 1   |                                   |        |          |           |
 2   |                                   |        |          |           |
 3   |                                   |        |          |           |
 4   |                                   |        |          |           |
 5   |                                   |        |          |           |
 6   |                                   |        |          |           |
 7   |                                   |        |          |           |
 8   |                                   |        |          |           |
 9   |                                   |        |          |           |
10   |                                   |        |          |           |
11   |                                   |        |          |           |
12   |                                   |        |          |           |
13   |                                   |        |          |           |
14   |                                   |        |          |           |
-----+-----------------------------------+--------+----------+-----------+
                                         |        |          |           |
                                         +--------+----------+-----------+
```

***** DELCHECK.CMD *****

Este fichero se usa para comprobar los artículos que se han servido contra los artículos del pedido original.

```
ERASE
SET TALK OFF
STORE "F" TO CHECK
STORE "             " TO MFIND
TEXT
    NECESITA INTRODUCIR EL NUMERO DE PEDIDO RELACIONADO CON LOS

    DETALLES DE LA ENTREGA

    INTRODUZCA NUMERO DE PEDIDO DE LA NOTA DE ENTREGA

ENDTEXT
@ 10,10 GET MFIND
READ
SELECT PRIMARY
USE B:ORDERS INDEX B:ORDIDX1
DO WHILE CHECK="F"
    FIND &MFIND
    IF #=0
         @ 10,10 SAY "                                             "
         @ 12,10 SAY "NO HAY PEDIDO REGISTRADO CON ESE NUMERO"
         @ 14,10 SAY "COMPROBAR NUMERO DE PEDIDO E INTRODUCIRLO DE NUEVO"
         @ 14,61 GET MFIND
         READ
    ELSE
         STORE "T" TO CHECK
    ENDIF
ENDDO
SELECT SECONDARY
USE B:GOODS
APPEND BLANK
DO GOODSENT
```

El fichero GOODSENT es una pantalla que usa las áreas de base de datos primaria y secundaria. La información del pedido viene del área primaria y los artículos recibidos se introducen en el área secundaria.

***** Fichero GOODSENT.ZPR *****

```
                    ** ENTRADA DE ARTICULOS RECIBIDOS **

--------------------------------------+------------------------------------
NUMERO PEDIDO  @ORD:NO                |NOTA DE ENTREGA  #DELNOTE
                                      |
PROVEEDOR      @S:CODE                |SERVIDO POR      #DELNOTE
--------------------------------------+--------------------+---------------
ARTICULOS PEDIDOS                     |SERVIDOS     S/N    |CANTIDAD
--------------------------------------+--------------------+---------------
 1 @ITEM1                     @QUANT1 | #DITEM1            |#DELQUA1
 2 @ITEM2                     @QUANT2 | #DITEM2            |#DELQUA2
 3 @ITEM3                     @QUANT3 | #DITEM3            |#DELQUA3
 4                                    |                    |
```

```
5                                    |                  |
6                                    |                  |
7                                    |                  |
8                                    |                  |
9                                    |                  |
10                                   |                  |
11                                   |                  |
12                                   |                  |
13                                   |                  |
14                                   |                  |
-------------------------------------+------------------+----------------
```

La pantalla ORDGREV se puede usar también para presentar el estado de un pedido con respecto a los artículos recibidos contra el pedido. Un sistema como este puede usar en el fichero de pantalla variables primarias y secundarias.

***** Fichero ORDGREV.ZPR *****

```
                    ** FACTURA DE ARTICULOS RECIBIDOS **

-------------------------------------+-------------------------------------
NUMERO PEDIDO                        |NOTA DE ENTREGA
                                     |
PROVEEDOR                            |SERVIDO POR
-------------------------------------+------------------+------------------
ARTICULOS PEDIDOS                    |SERVIDOS     S/N  |CANTIDAD
-------------------------------------+------------------+------------------
1                                    |                  |
2                                    |                  |
3                                    |                  |
4                                    |                  |
5                                    |                  |
6                                    |                  |
7                                    |                  |
8                                    |                  |
9                                    |                  |
10                                   |                  |
11                                   |                  |
12                                   |                  |
13                                   |                  |
14                                   |                  |
-------------------------------------+------------------+----------------
```

Los sistemas de este capítulo ilustran las distintas formas en que se pueden resolver los problemas usando dBaseII. No proporcionan necesariamente las solucione mejores o más simples. Para cada problema habrá por lo menos tantas soluciones como personas las puedan desarrollar, ya sea usando algunas de las técnicas mostradas aquí o desarrollando las suyas propias.

CAPITULO 9

Programas dBaseIII

En el sumario se indican los cambios en las especificaciones del dBaseIII. Este capítulo intentará identificar el efecto de alguno de estos cambios en programas escritos para dBaseII y las implicaciones de algunas de las nuevas facilidades en el diseño de programas.

El disco de utilidades del dBaseIII tiene un fichero llamado dConvert que realiza la mayoría del trabajo necesario para convertir programas de dBaseII al formato dBaseIII. Si su programa de aplicación está escrito en dBaseIII e intenta usar bases de datos, ficheros de índices y de informes existentes, el dConvert los cambiará por usted.

Para usar el dConvert en un sistema con dos unidades de disco, debe seguir el siguiente procedimiento:

1) Inserte el disco con el dConvert en la unidad A.

2) Introduzca el comando:

```
DCONVERT A: B:
```

3) saque el disco de la unidad A.

4) Inserte el primer disco de datos a convertir.

El formato del comando dConvert indica qué unidades se usan como origen y destino:

```
DCONVERT <unidad origen> <unidad destino>
```

Si las unidades de origen y destino son la misma, teclee DCONVERT sin ninguna letra de unidad:

```
A> DCONVERT
```

Cuando arranca el dConvert, aparece en la pantalla un menú con todos los tipos de ficheros posibles, una opción de ayuda y otra de salida. Para seleccionar una opción del menú indíquela y pulse la tecla RETURN. En ese momento aparecen todos los ficheros existentes del tipo indicado, junto con la cantidad de espacio disponible en el disco. Se introduce el nombre del fichero y dConvert realiza la conversión.

Es posible usar dConvert directamente desde el sistema, pero en este caso se debe especificar el tipo de los ficheros:

```
A> DCONVERT A:MIFICH.DBF B:MIFICH.DBF
```

Si su sistema tiene disco duro y quiere colocar sus ficheros convertidos en el disco C, la línea de comando debe ser:

```
A> DCONVERT A:MIFICH.DBF C:MIFICH.DBF
```

Si el fichero a convertir es muy largo, tardará algún tiempo, así que dConvert imprimirá un punto cada pocos segundos para indicarle que todavía está trabajando. Una vez completado el trabajo, renombra la base de datos original de dBaseII cambiando la última letra de la extensión por una B.

```
MIFICH.DBF A MIFICH.DBB
```

Efecto de dConvert en ficheros no de programa

Ficheros de Bases de Datos
Los ficheros dBaseII se convierten completamente al formato dBaseIII. Los dos puntos en los nombres de los campos se convierten en caracteres de subrayado(_). Los ficheros dBaseIII se pueden convertir al formato dBaseII, suponiendo que estén dentro de los límites de las bases de datos dBaseII.

Ficheros Indice
Cuando se usa el dConvert con ficheros índice, crea un fichero de comandos con el mismo nombre del fichero índice y la extensión .RX. El fichero índice se crea en dBaseIII usando el fichero de datos y ejecutando el fichero de comandos generado.

Por ejemplo, si la base de datos de dBaseII tiene el nombre NOMBRES.DBF y un índice NOMBRES.IDX, para crear este fichero de índice en dBaseIII después de usar el dConvert:

```
. USE NOMBRES
. DO NOMBRES.RX
```

Se asume que la base de datos NOMBRES ya ha sido convertida.

Ficheros de formato de informe
Los ficheros de formato de informe en dBaseIII no están en formato ASCII, dConvert cambiará los formularios de informes creados en dBaseII. Si se han usado algunos comandos dBaseII, dConvert los cambiará también.

Ficheros de memoria
Los ficheros de memoria son completamente convertidos a formato dBaseIII; sin embargo, los ficheros serán aproximadamente un 25% más largos.

Ficheros de comandos
La conversión de ficheros de comandos no es completa debido a los cambios significativos del sistema entre dBaseII y dBaseIII. Cuando haya un problema, dConvert añadirá comentarios. Un procedimiento útil cuando se está trabajando con ficheros de comandos es imprimir los listados de dBaseII y el producido por dConvert. Esto permite analizar rápidamente las áreas donde puede haber problemas ya que los comentarios van precedidos por '*!!'. dConvert incluye automáticamente un comentario en el fichero convertido, que indica el número de la versión de dConvert usada. Se añaden algunos comandos adicionales:

SET HEADING OFF   desactiva la aparición de los nombres de los campos en los comandos LIST y DISPLAY que produce el dBaseIII

SET SAFETY OFF   desactiva los mensajes de aviso antes de reescribir un fichero

Las principales áreas que no puede procesar el dConvert automáticamente son:

1) Variables en memoria

2) Apertura de ficheros múltiples

En el caso de las variables en memoria, el cambio de estructura en dBaseIII hace que, a menos que la variable sea declarada PUBLIC, exista solamente durante el programa que la ha creado. Existen muchos ejemplo de programas en dBaseII que usan variables en memoria en un programa para preparar acciones en otro al que llama. Un ejemplo de esto es el uso de una variable en memoria creada durante un proceso de actualización que hace que se muestre la opción de mantenimiento en el menú.

(1)

MENU (sin mantenimiento de ficheros)   UPDATE.PRG

(2)

PREPARA VARIABLE EN MEMORIA

(3)--------------------------------

(1) El programa del menú llama a la rutina de actualización

(2) Se crean nuevos datos, así que se prepara la variable en memoria que hace que se muestre la opción de mantenimiento de ficheros en el menú.

(3) A la vuelta al programa del menú, se muestra la opción que permite el mantenimiento de ficheros, ya que el cambio en el valor de la variable en memoria ha pasado de un programa al otro.

En dBaseII funcionará esta estrategia cuando la variable es creada por el programa de menú o el de actualización. En dBaseIII se debe definir la variable como PUBLIC para permitir su uso en todos los ficheros .PRG.

Los programas que usan bases de datos múltiples requieren ahora que se direccionen los campos por un seudónimo y el nombre de campo.

dBaseII

```
REPLACE P.NOMBRE WITH S.NOMBRE1
```

dBaseIII

```
REPLACE VIE->NOMBRE WITH NUE->NOMBRE
```

donde P y S son la identificación de las bases de datos primaria y

secundaria en dBaseII, y VIE y NUE son los seudónimos usados por dBaseIII. Este cambio se debe hacer manualmente, ya que ahora tenemos disponibles más de dos bases de datos.

Los comandos que han desaparecido del dBaseII al dBaseIII se deben sustituir por los comandos adecuados.

Por último, debido al cambio de sintaxis en las constantes lógicas, dConvert cambiará "T" por ".T."; estos se deben volver a cambiar a sus valores originales.

Como ejemplo listamos un pequeño programa en dBaseII junto con su equivalente en dBaseIII obtenido con dConvert. El fichero que se va a convertir está en la unidad C y el convertido irá a parar a la unidad A. El comando

```
A> DCONVERT C: A:
```

pondra a la unidad C como unidad de origen por defecto y a la unidad A como unidad de destino. Después de introducir el comando, aparece la pantalla del menú:

```
                dBASE II --> dBASE III

           1 - Database File        <.DBF>
           2 - Memory Variable File <.MEM>
           3 - Report Format File   <.FRM>
           4 - Command File         <.PRG>
           5 - Screen Format File   <.FMT>
           6 - Index File Help      <.NDX>
           7 - Un-dCONVERT III->II  <.DBF>

           9 -       Instructions
           0 -           EXIT

< Use cursor arrows to move between choices;  hit RETURN to select choice >
```

Seleccionando la opción 4, aparece un mensaje pidiendo el nombre del fichero a convertir.

```
Convert a command file

Enter input filename or RETURN to Exit:
```

El fichero de comandos será CONVERT.PRG, que se lista más adelante. Cuando dConvert está trabajando, aparece el mensaje WORKING, mas una serie de puntos. Después de la conversión, A:CONVERT.PRG se renombra como A:CONVERT.PRB. Se lista el fichero convertido para poder comparar.

Código fuente dBaseII para CONVERT.PRG

```
ERASE
SET TALK OFF
SELECT PRIMARY
USE NOMBRES INDEX IDX1
SELECT SECONDARY
USE OCUP INDEX NOMBS1
```

```
ACCEPT "INTRODUZCA EL NOMBRE DEL EMPLEADO " TO NOMBRE1
ERASE
@ 10,10 SAY "NOMBRE "
@ 12,10 SAY "DIRECCION   CALLE  "
@ 14,10 SAY "            CIUDAD "
@ 16,10 SAY "OCUPACION          "

SELECT PRIMARY
DO WHILE .NOT. EOF
    FIND &NOMBRE1

    @ 10,32 SAY NOMBRE
    @ 12,32 SAY DIR1
    @ 14,32 SAY DIR2

ENDDO

SELECT SECONDARY

DO WHILE .NOT. EOF

    FIND &NOMBRE1

    @ 16,32 SAY OCUP
ENDDO
```

El programa que hemos listado no es un ejemplo práctico, se ha construido solamente para ilustrar el resultado de la conversión de bases de datos múltiples y de la función EOF. El resultado de la conversión se lista a continuación.

Código fuente de CONVERT.PRG en dBaseIII:

```
*!!* dBASE CONVERT - dBASE III File Conversion Aid  v1.22 07/25/85
*
SET HEADING OFF
SET SAFETY OFF
CLEAR
SET TALK OFF
SELECT A
USE NOMBRES INDEX IDX1
SELECT B
USE OCUP INDEX NOMBS1

*!! There will be no automatic colon following this prompt string.
ACCEPT "INTRODUZCA EL NOMBRE DEL EMPLEADO " TO NOMBRE1
CLEAR
@ 10,10 SAY "NOMBRE "
@ 12,10 SAY "DIRECCION   CALLE  "
@ 14,10 SAY "            CIUDAD "
@ 16,10 SAY "OCUPACION          "

SELECT A
DO WHILE .NOT. EOF()
*!! EOF() will be true if NO FIND, and RECNO() will equal BOTTOM, not 0.
    FIND &NOMBRE1

    @ 10,32 SAY NOMBRE
```

```
   @ 12,32 SAY DIR1
   @ 14,32 SAY DIR2

ENDDO

SELECT B

DO WHILE .NOT. EOF()

*!! EOF() will be true if NO FIND, and RECNO() will equal BOTTOM, not 0.
   FIND &NOMBRE1

   @ 16,32 SAY OCUP
ENDDO
```

dConvert ha insertado comandos al principio y mensajes a lo largo del fichero, en los puntos donde los cambios en la sintaxis pueden causar dificultad.

Uso del dFormat para generar programas de pantalla

A primera vista, la inclusión del dFormat que viene en el paquete del dBaseIII es una ventaja significativa para la versión del dBase de 16 bits. dFormat produce un programa dBase con la extensión FMT, fichero de formato. Este tipo de fichero se usa normalmente en edición de pantalla completa o para añadir registros, y se le llama usando el siguiente comando:

```
SET FORMAT TO <nombre del fichero>
```

El dFormat es más restrictivo que el ZIP, ya que este último da la opción de generar ficheros de formato o de comandos. En algunos de los ejemplos que hemos visto en el capítulo 5, en los que hablamos de los comandos de pantalla, dijimos que podría ser útil una pantalla que constara solamente de comandos SAY. Para conseguirla con el dFormat, diseñe la pantalla sin sentencias GET y salve el resultado como FMT. Usando el sistema operativo, renombre el fichero como PRG; a partir de ahora se le puede llamar como un simple fichero de comandos. Se puede usar este procedimiento tanto con campo de una base de datos activa, como con variables en memoria. Si se usan variables en memoria, deben ser declaradas PUBLIC o usadas en una lista de parámetros, de forma que se puedan pasar cuando se ejecute el fichero de pantalla. Por ejemplo, consideremos el problema de usar una base de datos para mantener una lista de nombres, direcciones y nombres de compañías. El diseño podría ser:

```
NOMBRE                          <NOMBRE

DIRECCION   CALLE               <DIR1
            CIUDAD              <DIR2
            PROVINCIA           <DIR3
            CODIGO POSTAL       <PCODE

COMPAÑIA                        <COMPNAME
```

Usando dFormat se construiría un fichero de pantalla llamado LIST.

FMT, que contendría los siguientes comandos.

```
@ 2,10 SAY "NOMBRE"
@ 2,40 GET NOMBRE
@ 4,10 SAY "DIRECCION   CALLE"
@ 4,40 GET DIR1
@ 5,22 SAY "CIUDAD"
@ 5,40 GET DIR2
@ 6,22 SAY "PROVINCIA"
@ 6,40 GET DIR3
@ 7,22 SAY "CODIGO POSTAL"
@ 7,40 GET PCODE
@ 9,10 SAY "COMPAÑIA"
@ 9,40 GET COMPNAME
```

Usando este fichero, se puede escribir una rutina simple de entrada de datos:

```
USE NOMBRES
SET FORMAT TO LIST
APPEND BLANK
READ

etc.
```

Este programa nos permitirá añadir un registro. Para permitir añadir continuamente, se debe usar un bucle DO WHILE:

```
USE NOMBRES
CONT="S"
SET FORMAT TO LIST

DO WHILE CONT="S"
    APPEND BLANK
    READ

    @ 18,10 " ¿AÑADIMOS MAS?  S/N"
    @ 18,40 GET CONT
    READ
    IF CONT="S"
         @ 18,10 CLEAR
    ENDIF
ENDDO
```

Observe el uso de CLEAR para terminar el comando de pantalla y borrar los mensajes. En este punto el programa es esencialmente el mismo que en dBaseII. Si se llama a este programa desde el menú, la variable CONT debe existir solamente dentro del fichero de comandos ENTER.PRG. Si CONT fuera declarada PUBLIC, existiría durante todos los programas. Previamente era necesario salir del dBase y usar un proceso de textos que permita incluir ficheros externos dentro del fichero actual. El editor incluido en el dBaseIII incluye esta facilidad, así que el fichero de pantalla creado con dFormat se puede leer dentro del fichero actual usando este editor.

El nuevo tipo de datos DATE simplifica el uso de las fechas en programación. Consideremos un ejemplo simple: si se manda una cuenta en FECHA1, ¿qué fechas se deben usar para letras a 30, 60 y 90 días?

Asumimos que la fecha del sistema es DATE y que se pone al principio del programa.

```
STORE DATE() TO FECHA1
STORE DATE()+30 TO FECHA2
STORE DATE()+60 TO FECHA3
STORE DATE()+90 TO FECHA4
```

FECHA2, FECHA3 y FECHA4 son las fechas en las que se deben mandar las letras a 30, 60 y 90 días. Así, si comprobamos una fecha en particular, se seleccionarán todos los registros para los que DATE() sea igual al campo requerido.

Facilidades adicionales permiten introducir días y meses para fechas determinadas:

```
? CMONTH(DATE())          DATE()=30/10/84

. SEPTIEMBRE
```

Se puede usar en una forma estándar para obtener el mes de dentro de 50 días:

```
? CMONTH(DATE()+50)

. NOVIEMBRE
```

Este comando se puede usar para introducir el mes en documentos impresos:

```
STORE CMONTH(DATE()+50) TO MES
? "SE REQUIERE EL ENVIO ANTES DE "+MES
```

Además se puede usar la función YEAR para proporcionar a la impresora o pantalla una salida que contenga el mes y el año:

```
STORE YEAR(DATE()+50) TO MAÑO
? "SE REQUIERE EL ENVIO ANTES DE "+MES+"/"+MAÑO
```

Hay muchas aplicaciones para estas funciones, que se irán descubriendo con el uso. Se debe tener en cuenta que, como las fechas están en formato Juliano, se pueden realizar sumas y restas. Esta facilidad puede ser particularmente útil en programas para reservas de todo tipo. Las fechas que se usaban en dBaseII estaban en formato de carácter, pero se podían convertir en formato Juliano. La comprobación de validez de fechas ha sido siempre un problema que ha desaparecido ahora. Una aplicación potencial podría ser el determinar la fecha final de una reserva de hotel:

```
REGISTRADO EN 02/10/86
DURACION DE ESTANCIA 14 DIAS
```

¿Cuál es la fecha del último día de estancia? En dBaseII involucraba un gran número de pasos de programación, pero en dBaseIII se elimina esta dificultad:

```
STORE CDAY(DATE()) TO PRINCIP
STORE CDAY(DATE()+14) TO FINAL
```

```
STORE (DATE()+14) TO FECHAF
```

Ahora PRINCIP es el día, en caracteres, del principio de la reserva. FINAL es el último día de la reserva. FECHAF es la fecha en la que termina la reserva. Los valores de estas variables se pueden usar, como se ha visto antes, en la salida formateada, ej. para generar cartas o facturas.

En dBaseII no era posible convertir de carácter a numérico: esto se ha corregido en dBaseIII. Como ya se ha mencionado, las variables en memoria en dBaseIII existen solamente dentro del programa que las crea, a menos que se declaren PUBLIC. Esto nos da la ventaja de que en aplicaciones grandes, en las que intervienen varios programadores, no estarán limitados a unas variables concretas. Los programas pueden escribirse como módulos con sus propias variables. En esta situación, se debe pasar la información de un programa a otro, según se necesite. dBaseIII permite el paso de parámetros. Un ejemplo puede ser el programa de factura de producción del Capítulo 8. Para producir la factura se requiere una fecha de comienzo y una de terminación. La impresión de la factura se puede hacer en diferentes programas:

```
DO GETDATES

DO BILLPRNT
```

El programa GETDATES lee el registro correcto con las fechas de comienzo y terminación del periodo facturable. Esta información se debe pasar al programa de impresión de facturas. El programa BILLPRNT tiene una sentencia PARAMETERS identificando las variables que se le deben pasar para la ejecución.

El uso de esta técnica es nueva en dBase y su implementación requiere un rediseño significativo de los programas existentes. Probablemente, si los programas existentes funcionan correctamente, se deberían dejar sin tocar, y usar el sistema modular en el diseño de nuevos programas. La gran ventaja de este sistema es la posibilidad de usar los módulos varias veces en distintas situaciones, asumiendo que los nombres de los parámetros sean compatibles.

Finalmente, ahora se puede realizar la conversión de variables de caracteres a numéricas o viceversa:

```
STORE "1" TO CODE
LIST MEMORY
CODE     (C)   1

STORE VAL(CODE) TO CODE
LIST MEMORY
CODE     (N)   1
```

o

```
STORE 1 TO CODE1
LIST MEMORY
CODE1    (N)   1
STORE STR(CODE1,1,1) TO CODE1
LIST MEMORY
```

Cantidades nulas

La introducción de variables nulas crea una situación en la que es posible tener una cadena de longitud cero. LEN() devolverá un valor de 0 cuando se aplique a una cadena nula. Si el comandos ACCEPT o WAIT tiene como respuesta un retorno de carro, creará una variable nula. Es normal en dBaseII usar una técnica de programación que comprueba una respuesta comparándola con un espacio o una función LEN que de resultado 1. Esta comprobación no será posible en dBaseIII, ya que LEN(retorno de carro) es 0. En los programas existentes será necesario revisar la aplicación para descubrir si este cambio tiene efectos importantes. Las nuevas aplicaciones se deben escribir teniéndolo en cuenta.

Ficheros de índice

En dBaseII, el número de registro se pone a cero después de un FIND sin éxito. Para prevenir el mensaje de error y comprobar esta situación, el programador puede usar:

```
IF #=0
   @ 10,10 SAY " REGISTRO NO ENCONTRADO "
ELSE
   @ 10,10 SAY " REGISTRO ENCONTRADO    "
ENDIF
```

En dBaseIII se ha cambiado, de forma que si no se encuentra el registro, el número de registro se pone con el número del último registro del fichero y la función EOF se pone como verdadera. Esto es igual en la función LOCATE. Observe, sin embargo, que cuando se usa un fichero índice, el final del fichero no es necesariamente el último registro del mismo, por número de registro.El final del fichero es el último registro en el orden del índice. Una comprobación adecuada para un FIND puede ser:

```
IF EOF
   @ 10,10 SAY " REGISTRO NO ENCONTRADO "
ELSE
   @ 10,10 SAY " REGISTRO ENCONTRADO    "
ENDIF
```

dConvert no realiza este cambio, ya que requiere un cambio en la lógica del programa.

Corregir y volver a introducir

En dBaseII, cuando se cometía un error de sintaxis u otro error, aparecía un mensaje junto con la línea que contenía el error. Después de esto, se daba la opción de corregir y continuar el programa. Las correcciones introducidas de esta forma no eran permanentes. Esta fa-

cilidad ha sido eliminada en dBaseIII y reemplazada por otra que ofrece una ayuda. A primera vista puede parecer que es una pérdida; sin embargo, pensemos en el número de veces que se usaba esta facilidad para corregir un error y luego se olvidaba hasta la siguiente vez. Tenga en cuenta que en dBaseIII, SET STEP ON no permite introducir nada más desde el teclado.

Tecla ESC

En dBaseII, esta tecla se podía usar para abortar un programa. En dBaseIII, la tecla ESC hace que se pare el programa y espere una respuesta. La respuesta puede ser, o que pare el programa definitivamente, o que siga ejecutándolo. Un uso particularmente útil de esta facilidad es cuando se atasca el papel en la impresora. La tecla ESC actúa en este caso para detener la impresión mientras se ajusta el papel, permitiendo al programa seguir ejecutándose después.

Para parar la ejecución de un fichero de comandos se debe usar el comando EXIT. EXIT devuelve control al programa que lo ha llamado. Esto se puede usar como forma de salida desde un programa de entrada de datos a uno de presentación, de la misma forma que se usaba RETURN en dBaseII:

```
@ 10,10 SAY " INTRODUZCA UN NOMBRE (Q PARA SALIR)"
@ 10,10 GET NOMBRE
IF NOMBRE="Q"
   EXIT
ENDIF
```

En dBaseII se puede programar la misma función así:

```
@ 10,10 SAY " INTRODUZCA UN NOMBRE (Q PARA SALIR)"
@ 10,10 GET NOMBRE
IF NOMBRE="Q"
   RETURN
ENDIF
```

En sistemas grandes que han sido diseñados para usar estructuras de tipo árbol, se proporciona un comando RETURN TO MASTER. Este comando fuerza a retornar el control al primer programa llamado desde el dBase o el sistema operativo. Un uso típico de este comando sería el tratamiento de la opción de retorno desde el sistema a un menú principal:

```
1) MENU PRINCIPAL
2)
3)

etc.

INTRODUCIR ACTION
@ 20,20 GET ACTION
READ

DO CASE
   CASE ACTION="1"
```

```
        RETURN TO MASTER
    CASE ACTION="2"
        DO PRINT
    CASE ACTION="3"
        DO otro programa
```

etc.

Conexión de ficheros y uso de ficheros múltiples

La introducción de los nombres ALIAS (seudónimos) para los ficheros múltiples, y la opción de tener hasta 10 ficheros abiertos al mismo tiempo, crea nuevos problemas. Los programas existentes usarán las opciones PRIMARY y SECONDARY del dBaseII. dConvert cambiará SELECT PRIMARY o SELECT SECONDARY en SELECT 1 o SELECT 2; no puede aplicar los ALIAS correctos. Si embargo, usando un buen procesador de textos nos permitirá la sustitución múltiple de los comandos usando P.NOMBRE y S.NOMBRE a la convención de dBaseIII ALIAS->NOMBRE. Recuerde cambiar las sentencias USE para reflejar los ALIAS elegidos.

SET RELATION TO se usa para conectar bases de datos de diferentes áreas de trabajo. El uso de este comando hace que el apuntador de registros en la bases de datos identificada se mantenga en relación a la base de datos activa. Tenga en cuenta que solo se puede poner una relación desde cualquiera de las 10 áreas de trabajo que pueden existir.

Sumario

Los programas que ya funcionaban en dBaseII se pueden convertir para ejecutarlos en dBaseIII mediante el programa dConvert y, cuando sea necesario, realizando cambios adicionales.

Las nuevas aplicaciones escritas en dBaseIII pueden usar técnicas similares, suponiendo que se tengan en cuenta los cambios en las funciones y el efecto de condiciones como 'NO ENCONTRADO'. Aunque los diseños de programas desarrollados en dBaseII se pueden pasar a dBaseIII, esto ignoraría los cambios hechos en dBaseIII. El principal efecto de estos cambios es hacer al dBaseIII que sea capaz de usar un sistema de programación modular que permita usar los módulos en otras aplicaciones. La eficacia de sistema compensa el tiempo y esfuerzo requeridos en la fase de diseño de una aplicación.

APENDICE 1

Ficheros de Utilidad del dBaseII

Los ficheros esenciales de dBaseII son:

DBASE.COM	- programa principal del sistema
DBASEOVR.COM	- fichero de recubrimiento y mensajes del sistema.
DBASEMSG.TXT	- fichero de AYUDA (no se necesita si no usa HELP)
INSTALL.COM	- Programa de instalación (no se necesita después de haberlo instalado)

Los demás ficheros que vienen en el disco (o en otros discos incluidos con el sistema) no son esenciales, pero son útiles. Se describen a continuación:

Utilidades

Los ficheros de utilidad del dBaseII son:

STARTUP.CMD	Este fichero de comandos le ayudará a determinar si su terminal está instalado correctamente. Teclee DO STARTUP después de entrar en dBase.
SETS.CMD	Este fichero de comandos restaura el dBase a todos los valores por defecto de los parámetros SET. Usted puede modificar este fichero para que restaure su propia lista de parámetros.
DATESYS.CMD	Este fichero de comandos llama a una rutina en lenguaje ensamblador que comprueba la validez de la fecha, y pone después la fecha del sistema dBase.
LABELS.CMD	Este fichero de comandos imprime etiquetas para el correo.
NAMES.DBF	Base de datos usada por LABELS.CMD.
CREATE.CMD	Este fichero de comandos crea otro fichero de comandos con la documentación usual y la sugerida para un fichero de comandos.
DATER.CMD	Este fichero de comandos demuestra la conversión entre la fecha del calendario y la Juliana.
DATETEST.HEX	Rutina rápida de comprobación de fechas, en lenguaje ensamblador, que es llamada por varios programas de ejemplo.

ZIP.COM	Un programa ejecutable desde CP/M que acelera el desarrollo de pantallas para entrada y presentación de datos.
ZIPIN.COM	Programa de instalación del ZIP; ejecute este programa si el terminal no aparece correctamente cuando ejecute el ZIP.
ZSCRN.OVL	Parte de ZIP.
DGEN.OVL	Parte de ZIP que genera los ficheros dBase.

Instalación

Para poder usar la facilidad del dBaseII de pantalla completa, su terminal debe ser instalado correctamente. Muchos formatos dBase está preinstalados. Si el suyo no lo está, su pantalla aparecerá distorsionada cuando realice operaciones de pantalla completa como APPEND o EDIT. Para corregirlo, debe instalar el dBaseII ejecutando el comandos INSTALL.COM, llamándolo desde el sistema como INSTALL. Ya que puede diagnosticar el estado de su terminal en cualquier momento usando STARTUP.CMD (llamándolo desde el dBase con DO STARTUP), puede usarlo antes de instalar el dBase. El le dirá si necesita ejecutar el programa de instalación, y puede usarlo después para comprobar los resultados.

Facilidad de ayuda del dBaseII

A continuación le damos una lista de los comandos de dBaseII. Este fichero HELP contiene una breve descripción de cada comando, así como un ejemplo del comando con su sintaxis correcta. Recuerde, si embargo, que el fichero HELP es limitado, y solo es útil para reducir el número de referencias que tendrá que hacer al manual del dBaseII.

Para acceder a la entrada de un comando de dBaseII (o a otra entrada del fichero HELP), responda el mensaje del dBaseII "." con HELP <nombre del comando> y RETURN. Ej. HELP CREATE . La información deseada aparecerá y después volverá al mensaje ".", de forma que pueda seguir trabajando. En algunos casos la entrada ocupará la siguiente pantalla de información, solo debe teclear cualquier carácter al mensaje "WAITING". Puede terminar el mensaje de ayuda pulsando ESC dos veces.

>>>>>>>>>> Teclee 'HELP dBase' para otra información importante

Resumen de los comandos disponibles en HELP:

?	Muestra una expresión, variable o campo.
??	Muestra una lista de expresiones sin precederla con una alimentación de línea.
@	Muestra en la pantalla o impresora, datos formateados por el usuario.

ACCEPT	Permite introducir cadenas de caracteres dentro de variables de memoria.
APPEND	Añade información desde otra base de datos dBaseII o ficheros en formato DELIMITED o del sistema.
BROWSE	Visión y edición de una base de datos a pantalla completa.
CANCEL	Cancela la ejecución de un fichero de comandos.
CHANGE	Edita campos de una base de datos.
CLEAR	Cierra las bases de datos en uso y libera las variables en memoria.
CONTINUE	Continúa la acción de búsqueda del comando LOCATE.
COPY	Crea una copia de una base de datos existente.
COUNT	Cuenta el número de registros de un fichero que siguen un criterio concreto.
CREATE	Crea una base de datos nueva.
DELETE	Borra un fichero o marca registros para borrar.
DISPLAY	Presenta listados de ficheros, registros o estructura de bases de datos, variables en memoria o estado.
DO	Ejecuta ficheros de comandos o bucles estructurados en ficheros de comandos.
EDIT	Inicia la edición de registros en las bases de datos.
EJECT	Hace saltar una página a la impresora.
ELSE	Paso alternativo en la ejecución de un comando dentro de una estructura IF.
ENDCASE	Termina un comando CASE.
ENDDO	Termina un comando DO WHILE,
ENDIF	Termina un comando IF.
ENDTEXT	Termina un comando TEXT.
ERASE	Limpia la pantalla.
FIND	Se posiciona en el registro correspondiente a la clave en un fichero indexado.
GO o GOTO	Se coloca en un registro o posición específica dentro de un fichero.
HELP	Accede al fichero HELP.
IF	Permite la ejecución condicional de comandos.
INDEX	Crea un fichero de índice.
INPUT	Permite introducir expresiones en variable en memoria.
INSERT	Inserta un nuevo registro en una base de datos.
JOIN	Obtiene como resultado la unión de dos bases de datos.
LIST	Lista ficheros, registros o estructuras de bases de datos, y estados.
LOCATE	Busca un registro que cumpla una condición.
LOOP	Salta al principio de un comando DO WHILE.
MODIFY	Crea o edita un fichero de comandos o modifica la estructura de una base de datos.
NOTE o *	Permite insertar comentarios en un fichero de comandos.
PACK	Borra los registros marcados para borrar.
QUIT	Sale del dBase y vuelve al sistema operativo.
READ	Inicia la edición de pantalla completa aceptando las entradas dentro de las variables a las que se accede mediante @ GET.
RECALL	Elimina las marcas de borrado.
REINDEX	Actualiza un fichero de índice existente.
RELEASE	Elimina las variables en memoria no deseadas y libera espacio en memoria.
REMARK	Permite mostrar cualquier carácter..
RENAME	renombra un fichero.

REPLACE	Cambia la información en un registro por campo.
REPORT	Formatea y muestra un informe.
RESET	Hace saber al sistema operativo que se ha cambiado un disco.
RESTORE	Carga las variables de memoria almacenadas en ficheros MEM.
RETURN	Termina un fichero de comandos.
SAVE	Copia las variables en memoria a un fichero en disco.
SELECT	Cambia entre ficheros en las áreas de trabajo PRIMARY y SECONDARY.
SET	Pone los parámetros de control del dBase.
SKIP	Se posiciona en la base de datos.
SORT	Crea una copia de la base de datos, ordenada por uno de sus campos.
STORE	Crea variables en memoria.
SUM	Realiza y muestra la suma de campos de una base de datos.
TEXT	Permite escribir en pantalla un bloque de texto dentro de una base de datos.
TOTAL	Crea una copia resumen de bases de datos combinando la información de determinados campos que sigan ciertos criterios.
UPDATE	Permite actualizar una base de datos.
USE	Especifica la base de datos que se va a usar hasta el siguiente comando USE.
WAIT	suspende el proceso del fichero de comandos hasta que recibe una entrada del usuario.

Otras palabras clave son:

UTILITIES	FULL SCREEN	LIMITS	BACKUP	
INSTALL	NEW	ERRORS	CP/M	
EXAMPLES	FUNCTIONS	DBASE	HELP	RUNTIME

REINDEX	Alinea y actualiza un fichero de índice existente.
TEXT	muestra líneas múltiples de texto, terminando con ENDTEXT.
RANK()	Función que devuelve el valor ASCII del primer carácter.

Comandos con suplemento:

APPEND,INSERT,EDIT,CREATE	Puede usar un fichero de formato para controlar la pantalla.
BROWSE	Se le puede dar una lista de campos con los que trabajar.
DISPLAY STATUS	Muestra el estado de las bases de datos y los índices en uso y de los parámetros puestos con SET.
READ y REPLACE	Se pueden direccionar para que no realicen acceso a disco cuando cambian datos que no tienen clave.
RELEASE	Libera grupos de variables en memoria.
RESTORE	restaura variables desde un fichero, añadiéndolas a las que ya hay en memoria.
SAVE	Se pueden salvar grupos de variables.
SET RAW ON	Elimina espacios extra en los comandos DISPLAY y ?
UPDATE	La cláusula REPLACE puede tener frases WITH.

Funciones dBaseII

@	@(<ccadena1>,<ccadena2>) -- función AT. Da un entero cuyo valor es el número del carácter en <ccadena2> en el que empieza una subcadena igual a <ccadena1>.
*	Función de registro borrado, se calcula como verdadero si el registro actual está marcado para borrar.
#	Función de número de registro. Da el valor del entero correspondiente al número del registro actual.
!	!(<ccadena>) - Función de mayúsculas. pone <ccadena> en mayúsculas.
$	$(<ccadena>,<empiece>,<longitud>) - Función de subcadena. forma una cadena de caracteres desde la parte especificada de otra cadena.
CHR	CHR(<expresión numérica>) - Da el carácter ASCII equivalente a la <expresión numérica>. Ej. CHR(7) hace sonar la alarma.
DATE()	Devuelve una cadena de caracteres que contiene la fecha del sistema en formato xx/xx/xx.
EOF	Función de fin de fichero. Su resultado es verdadero (T) si se ha hecho un intento de ir más allá del último registro de la base de datos.
FILE	FILE(<fichero>) - Su resultado es verdadero (T) si el <fichero> existe en la unidad por defecto, y falso (F) si no existe.
INT	INT (<expresión numérica>) Función de entero. Redondea un número para obtener el número entero más próximo.
LEN	LEN(<ccadena>) - Función de longitud. Devuelve el número de caracteres en <ccadena>. ? LEN('HOLA') = 4
RANK	RANK(<ccadena>) - Devuelve el valor (número ASCII) del carácter de más a la izquierda en <ccadena>.
STR	STR(<expresión numérica>,<ancho> [,<decimales>]) - Función de cadena. convierte una expresión numérica en una cadena de caracteres.
TEST	TEST(<expr>) - Función para determinar si <expr> es válida o analizable. <expr> puede ser una expresión numérica, otra función, un nombre de campo, o cualquier combinación (pero no un comando dBase). Una <expr> válida da como resultado un 1.
TRIM	TRIM(<ccadena>) - Esta función elimina los blancos al final de <ccadena>. ? TRIM(PRIMER)+' '+ULTIMO.
TYPE	TYPE(<expr>) - Da como resultado una cadena de un carácter que contiene 'C','N','L' o 'U' si <expr> es de tipo Carácter, Numérico, Lógico o Indefinido.

VAL VAL(<ccadena>) - Función de valor. Convierte una cadena de caracteres numéricos en una expresión numérica.
? VAL('12345')

Mensajes de Error de dBaseII

BAD DECIMAL WIDTH FIELD
- ANCHO DE CAMPO DECIMAL INVALIDO
Reintroduzca la parte decimal de la definición del campo.

BAD FILE NAME
- NOMBRE DE FICHERO INVALIDO
Error de sintaxis en el nombre del fichero.

BAD NAME FIELD
- NOMBRE DE CAMPO INVALIDO
Redefina el nombre del campo en CREATE.

BAD TYPE FIELD
- TIPO DE CAMPO INVALIDO
Debe ser C(carácter), N(numérico) o L(lógico).

BAD WIDTH FIELD
- ANCHO DE CAMPO INVALIDO
Redefina el tamaño del campo de datos, debe estar entre 1 y 255.

*** BEYOND STRING
- FUERA DE LA CADENA
Reescriba la subcadena ($) con los parámetros corregidos.

CANNOT INSERT - THERE ARE NO RECORDS IN DATABASE FILE
- NO SE PUEDE INSERTAR - NO HAY REGISTROS EN LA BASE DE DATOS
Use el comando APPEND en este caso.

CANNOT OPEN FILE
- NO SE PUEDE ABRIR EL FICHERO
Compruebe la existencia o la integridad del fichero MEM o HEX.

COMMAND FILE CANNOT BE FOUND
- NO SE PUEDE ENCONTRAR EL FICHERO DE COMANDOS
Compruebe el nombre especificado o la unidad por defecto.

DATA ITEM NOT FOUND
- DATOS NO ENCONTRADOS
Reescriba el comando REPLACE, o compruebe la estructura del fichero para encontrar el nombre de campo correcto.

DATABASE IN USE NOT INDEXED
- LA BASE DE DATOS EN USO NO ESTA INDEXADA
Solo está permitido usar el comando FIND con una base de datos indexada.

DIRECTORY IS FULL
- EL DIRECTORIO ESTA LLENO
El directorio del disco no puede contener más ficheros.

DISK IS FULL
- EL DISCO ESTA LLENO
No queda espacio libre en el disco. Use el comando DELETE FILE para borrar algún fichero que no sea necesario.

END OF FILE FOUND UNEXPECTEDLY
- FIN DE FICHERO ENCONTRADO INESPERADAMENTE
La base de datos en uso no tiene el formato correcto. dBase no está seguro de que sea un fichero DBF.

"FIELD" PHRASE NOT FOUND
- PALABRA "FIELD" NO ENCONTRADA
Vuelva a escribir el comando CHANGE.

FILE ALREADY EXISTS
- YA EXISTE EL FICHERO
Borre el fichero innecesario antes de usar RENAME.

FILE DOES NOT EXISTS
- EL FICHERO NO EXISTE
Use DISPLAY FILE LIKE *.* para asegurarse de que existe el fichero.

FILE IS CURRENTLY OPEN
- EL FICHERO ESTA ACTUALMENTE ABIERTO
Teclee el comando USE o CLEAR para cerrar el fichero.

FORMAT FILE CANNOT BE OPENED
- EL FICHERO DE FORMATO NO SE PUEDE ABRIR
Compruebe la integridad del fichero .FMT.

FORMAT FILE HAS NOT BEEN SET
- NO SE HA PREPARADO EL FICHERO DE FORMATO
Ejecute un SET para el fichero .FMT adecuado.

ILLEGAL DATA TYPE SORT
- TIPO DE DATOS ILEGAL PARA CLASIFICAR
No se puede hacer un SORT por un campo lógico.

ILLEGAL GOTO VALUE
- VALOR DEL 'GOTO' ILEGAL
El registro debe estar en el rango de 0 a 65535.

ILLEGAL VARIABLE NAME
- NOMBRE DE VARIABLE ILEGAL
En los nombres de campos y de variables se permiten solamente caracteres alfabéticos, numéricos y el de dos puntos (:). Vuelva a definir la variable o el nombre del campo.

INDEX DOES NOT MATCH DATABASE
- EL INDICE NO COINCIDE CON AL BASE DE DATOS
No coincide la clave del índice con la base de datos. Inténtelo con otro fichero de índice.

INDEX FILE CANNOT BE OPENED
- EL FICHERO INDICE NO SE PUEDE ABRIR
Compruebe el nombre o indexe la base de datos.

JOIN ATTEMPTED TO GENERATE MORE THAN 65,534 RECORDS
- 'JOIN' INTENTO GENERAR MASA DE 65535 REGISTROS
La sentencia FOR ha permitido demasiados registros; hágala más restringida.

KEYS ARE NOT THE SAME LENGTH
- LAS CLAVES NO SON DE LA MISMA LONGITUD
El comando UPDATE requiere claves idénticas.

MACRO IS NOT A CHARACTER STRING
- MACRO NO ES UNA CADENA DE CARACTERES
La variable que va a ser expandida por una macro (&) debe ser una cadena de caracteres.

MORE THAN 5 FIELDS TO SUM
- MAS DE 5 CAMPOS A SUMAR
SUM está limitado a cinco campos al mismo tiempo.

MORE THAN 7 INDEX FILES SELECTED
- MAS DE 7 FICHEROS INDICE SELECCIONADOS
El número máximo de ficheros índice que se pueden abrir es 7. Si usa menos obtendrá mejores prestaciones.

NESTING LIMIT VIOLATION EXCEEDED
- EXCEDIDO EL LIMITE DE VIOLACION DE ANIDAMIENTO
No se pueden tener abiertos más de 16 ficheros de comandos al mismo tiempo.

NO EXPRESSION TO SUM
- NO HAY EXPRESION PARA SUMAR
El comando SUM necesita una expresión numérica para sumar.

NO "FOR" PHRASE
- NO HAY PALABRA 'FOR'
Vuelva a escribir el comando JOIN con la sintaxix correcta.

NO "FROM" PHRASE
- NO HAY PALABRA 'FROM'
Vuelva a escribir el comando UPDATE con la sintaxis correcta.

NO FIND
- NO ENCONTRADO
Es más un diagnostico que un mensaje de error. dBase no puede encontrar la clave. # se pone a 0.

NON-NUMERIC EXPRESSION
- EXPRESION NO NUMERICA
El comando SUM necesita una expresión numérica para sumar.

NOT A dBaseII DATABASE
- NO ES UNA BASE DE DATOS dBase
El fichero .DBF que se ha abierto no fué creado por dBase.

"ON" PHRASE NOT FOUND
- PALABRA 'ON' NO ENCONTRADA
Vuelva a escribir el comando UPDATE o INDEX con la sintaxis correcta.

OUT OF MEMORY FOR MEMORY VARIABLES
- MEMORIA DE VARIABLES LLENA
Reduzca el número o el tamaño de las variables en memoria.

RECORD LENGTH EXCEEDS MAXIMUM SIZE (OF 1000)
- LA LONGITUD DEL REGISTROS EXCEDE EL TAMAÑO MAXIMO (DE 1000)
Reduzca el tamaño de algunos campos o cree una segunda base de datos con una clave común.

RECORD NOT IN INDEX
- EL REGISTRO NO ESTA EN EL INDICE
El fichero de índice no sa ha actualizado después de añadir un registro. Vuelva a indexar el fichero.

RECORD OUT OF RANGE
- REGISTRO FUERA DE RANGO
Se ha llamado a un número de registro que es mayor que el número de registros que hay en la base de datos. El fichero de índice no está actualizado; vuelva a indexar el ficheros.

SORTER INTERNAL ERROR, NOTIFY SCDP
- ERROR INTERNO DEL 'SORT', NOTIFIQUE AL SCDP
Error interno, contacte con ASHTON-TATE para pedir soporte.

SOURCE AND DESTINATION DATA TYPES ARE DIFFERENT
- EL TIPO DE LOS DATOS DE ORIGEN Y DESTINO ES DIFERENTE
Compruebe que el tipo de ambos datos es igual.

*** SYNTAX ERROR ***
- *** ERROR DE SINTAXIS ***
dBase no entiende el comando.

SINTAX ERROR IN FORMAT SPECIFICATION
- ERROR DE SINTAXIS EN ESPECIFICACION DE FORMATO
Comando @ SAY GET PICTURE mal preparado.

SYNTAX ERROR, RE-ENTER
- ERROR DE SINTAXIS, VUELVA A INTRODUCIR
INPUT, ACCEPT y REPORT requieren entradas sintacticamente correctas. Puede estar esperando un tipo de datos diferente.

"TO" PHRASE NOT FOUND
- PALABRA 'TO' NO ENCONTRADA
Vuelva a escribir el comando con la sintaxis correcta.

TOO MANY CHARACTERS
- DEMASIADOS CARACTERES
Solo cuando no está en pantalla completa. Los datos introducidos exceden la longitud del campo.

TOO MANY FILES ARE OPEN
- DEMASIADOS FICHEROS ABIERTOS
Solo se pueden abrir 16 ficheros de cualquier tipo (Comandos, .FMT, .NDX) al mismo tiempo.

TOO MANY MEMORY VARIABLES
- DEMASIADAS VARIABLES EN MEMORIA
No puede haber más de 64 variables en memoria.

TOO MANY RETURNS ENCOUNTERED
- DEMASIADOS 'RETURN' ENCONTRADOS
Error probable de estructura de un fichero de comandos. Compruebe el número y situación de los 'RETURN'.

"WITH" PHRASE NOT FOUND
- PALABRA 'WITH' NO ENCONTRADA
Vuelva a escribir el comando REPLACE con la sintaxis correcta.

UNASSIGNED FILE NUMBER
- NUMERO DE FICHERO SIN ASIGNAR
Error interno, contacte con ASHTON-TATE para obtener soporte. Si se ha usado HELP, puede que haya perdido el fichero DBASEMSG.TXT de la unidad.

*** UNKNOWN COMMAND
- *** COMANDO DESCONOCIDO
Compruebe el comando, dBase no lo entiende.

VARIABLE CANNOT BE FOUND
- NO SE PUEDE ENCONTRAR LA VARIABLE
Hace falta crear la variable, o compruebe el nombre del campo en la estructura de la base de datos

*** ZERO DIVIDE
- *** DIVISION ENTRE CERO
Se ha intentado dividir una expresión numérica entre cero.

RUNTIME

dBase RunTime es un producto de Ashton-Tate diseñado para asistir en el desarrollo de aplicaciones, para comercializar paquetes de programas escritos en dBaseII.

dBase RunTime tiene la capacidad de criptografiar los programas en dBaseII y hacer que la aplicación se ejecute solamente como fue diseñada. Ashton-Tate publica un catálogo de aplicaciones existentes y proporciona a los diseñadores de aplicaciones, otras ayudas para la comercialización de sus programas.

Si usted desarrolla una aplicación usando dBaseII y la quiere comercializar, póngase en contacto con el coordinador del RunTime en Ashton-Tate para obtener la información de cómo proceder. Para obtener información de las aplicaciones que funcionan en su sistema dBaseII, pregunte al Application Marketing Referral Service.

LIMITACIONES

Número de campos por registro	32 máximo
Número de caracteres por registro	1000 máximo
Número de registros por base de datos	65535 máximo
Número de caracteres por cadena de caracteres	254 máximo
Exactitud de los campos numéricos	10 dígitos
Número más grande	1.8 x 1063 aproximadamente
Número más pequeño	1.0 x 10-63 aproximadamente
Número de variables en memoria	64 máximo
Número de caracteres por línea de comandos	254 máximo
Número de expresiones en el comando SUM	5 máximo
Número de caracteres en una cabecera de REPORT	254 máximo
Número de campos en REPORT	24 máximo
Número de caracteres en una clave de índice	99 máximo
Número de GET pendientes	64 máximo
Número de ficheros abiertos al mismo tiempo	16 máximo
Longitud de un fichero de comandos ejecutable	ilimitada

APENDICE 2

Cambios en dBaseIII

dBaseIII está escrito en lenguaje C, excepto los interfases con el sistema operativo y las de entrada y salida. Usa 180 Kbytes de memoria, sin incluir la memoria requerida por el sistema operativo. Está diseñado para trabajar con DOS 2.0 y 2.1 en el PC de IBM y en sistemas que sean 100% compatibles. Se requiere un mínimo de 256 Kbytes de memoria, pero para poder usar el comando RUN hace falta más memoria. dBaseIII puede acceder al sistema de directorios en forma de árbol del DOS 2.0.

Estadísticas del dBaseIII

Número de campos por registro	128 máximo
Número de caracteres por registro	4000 máximo
Número de registros por base de datos	1000.000.000 máximo
Número más grande	1.0E+307 aproximadamente
Número más pequeño	1.0E-307 aproximadamente
Número de variables en memoria	256 máximo
Espacio para variables en memoria (en octetos)	6000 máximo
Número de ficheros abiertos al mismo tiempo	15 máximo
Número de bases de datos abiertas al mismo tiempo	10 máximo

CONFIG.DB

El fichero CONFIG.DB se puede usar para establecer el valor de la mayoría de los parámetros SET y de las teclas de función. El tamaño de la memoria se puede redefinir en CONFIG.DB. Si existe CONFIG.DB cuando se arranca el dBaseIII, se usan los valores de este fichero para reemplazar los valores en dBaseIII.

ALIAS

Ahora podemos referirnos a las bases de datos por un seudónimo (ALIAS) que se puede usar en lugar de los nombres, ej.:

```
USE PEDIDOOS ALIAS PED1
SELECT 2
USE FACTURAS ALIAS FAC1
SELECT PED1
DISP                          * Presenta el registro seleccionado
                              * de la base de datos ORDER
? FAC1->PROVEEDOR             * Imprime el valor actual del campo
                              * PROVEEDOR de la base de datos
                              * FACTURAS
```

Conexión de bases de datos

Para conectar dos bases de datos abiertas se usa el comandos SET RELATION TO:

```
SET RELATION TO PROVEEDOR INTO FACTURAS
```

Ahora, cuando se mueva el apuntador de registros en la base de datos PEDIDOS, el apuntador de registros en la base de datos FACTURAS se moverá al primer registro que coincida con el campo PROVEEDOR en la base de datos PEDIDOS.

Nuevos tipos de datos

MEMO

Un campo MEMO puede contener hasta 4000 caracteres, si se usa el procesador de textos del dBaseIII. Si se usa un procesador de textos estándar, el campo MEMO puede ser más grande. Si se edita una base de datos con un campo MEMO, aparecerá la palabra "MEMO" en la pantalla de edición. Mueva el cursor al campo MEMO y pulse ^PgDn. El procesador de textos del dBaseIII le mostrará el contenido del campo MEMO y le permitirá hacer cambios. La salida de campos MEMO se genera con los comandos DISPLAY y REPORT.

Restricciones en el uso de los campos MEMO:

Los campos MEMO no se pueden usar
- como claves en SORT.
- como claves en INDEX.
- en comandos de subcadena.
- en comandos de concatenación de cadenas.
- para ser accedidos por comandos @ SAY o @ GET.

DATE

El tipo de datos DATE permite cálculos internos usando fechas, pero muestra la fecha en la forma usual DD/MM/AA. Se han añadido varios comandos nuevos que permiten el manejo de este tipo de datos.

Comandos nuevos

MEMVAR=EXP

SUMA=1234 es lo mismo que STORE 1234 TO SUMA

ASSIST

Con esta opción se pueden usar por medio de un menús la mayoría de los comandos de dBaseIII, en formato de pantalla completa, ej. no es necesario memorizar los comandos.

AVERAGE

Calcula la media aritmética de las expresiones.

CLOSE ALTERNATE, DATABASES, FORMAT, INDEX O PROCEDURE

Cierra los ficheros del tipo elegido.

COPY FILE

Hace una copia de un fichero de cualquier tipo.

MODIFY LABEL

Permite crear y editar formularios de etiquetas, evitando la necesidad de escribir ficheros de comandos separados para esta función. Las etiquetas se imprimen usando el comando LABEL FORM.

PRIVATE [ALL [LIKE/EXCEPT <esqueleto>,]] [lista de variables]

Este comando oculta al programa de nivel superior, las variables en memoria identificadas.

PUBLIC [lista de variables]

Hace que las variables listadas estén disponibles en todos los programas.

RUN <comando>

Se usa para ejecutar dentro del dBaseIII, cualquier programa o comando de DOS.

SEEK

Este comando actúa como FIND, excepto que acepta una expresión como argumento.

SET

Es un programa de pantalla completa que permite al usuario cambiar la mayoría de los parámetros de control del dBaseIII desde un menú, en lugar de tener que introducirlos directamente.

SET COLOR TO <normal> [,<realzado>] [,<fondo>]

Este comando se usa para preparar la pantalla. Las condiciones pueden ser:

R = ROJO	U = Subrayado (para monocromo solo)
G = Verde	+ = Alta intensidad
B = Azul	* = Parpadeante
W = Blanco	! = Vídeo invertido

SET DECIMALS TO

Pone el mínimo número de cifras decimales que se mostrarán como resultado de una DIVISION, EXP(), LOG() y SQRT().

SET DELIMITERS

Permite seleccionar el carácter usado como delimitador de datos y cambia la presentación de los campos entre vídeo inverso o con el carácter especificado.

SET FILTER TO <condición>

Seleccionará solamente aquellos registros en la base de datos para los que la condición sea verdadera.

SET FIXED ON/OFF

Fija globalmente el número de caracteres decimales que se mostrarán con un SET DECIMALS TO.

SET FUNCTION (num tecla) TO "<cadena>"
Permite al usuario definir le cadena de caracteres que aparecerá como entrada cuando se pulse la tecla de función correspondiente.

SET FUNCTION 4 TO "DISP STRU"
Cuando se pulse la tecla de función 4, aparecerá la estructura de la base de datos en uso.

SET HEADING ON/OFF
Pone o quita la cabecera cuando se usan los comandos LIST o DISPLAY.

SET HELP ON/OFF
Si se pone ON, aparecerá el mensaje "¿Desea información de ayuda?" cuando se cometa algún error en modo interactivo.

SET MENU ON/OFF
Determina si aparece o no el menú de teclas de movimiento del cursor em los mandatos de pantalla completa.

SET PATH TO (vía 1),(vía 2)
Se usa para definir las vías alternativas que se pueden usar para encontrar los ficheros que no están en el directorio actual.

SET PROCEDURE TO (nombre fichero de procedimientos)
Abre el fichero de procedimientos especificado. Si tienen el mismo nombre un fichero de comandos y uno de procedimientos, se ejecutará antes el de comandos.

SET SAFETY ON/OFF
Si está puesto a ON, dBaseIII le pondrá un mensaje de aviso antes de escribir sobre un fichero, en caso contrario, escribirá sobre el fichero existente.

TYPE

Se usa para listar el contenido de un fichero en la pantalla o en la impresora. El nombre de fichero debe tener una extensión.

ZAP

Pone a cero el número de registros de la base de datos en uso y restaura todos los ficheros índice asociados. El mucho más rápido que DELETE ALL seguido de PACK.

Funciones nuevas

BOF()

Esta es la función de principio de fichero. La función se pone como verdadera (.T.) cuando se hace un intento de leer más allá del primer registro lógico de una base de datos activa. Se usa para evitar el error de fin de fichero cuando se está leyendo la base de datos en

orden inverso.

EXP

Esta función calcula el valor de un exponencial.

LOG

Calcula el logaritmo en base e de una expresión numérica (logaritmo natural).

ROUND

Redondea una expresión numérica al número especificado de caracteres decimales.

SQRT

Función para calcular raíces cuadradas de números positivos.

Se suministra un nuevo rango de funciones que actúan con variables de tipo DATE.

CTOD

Esta función convierte en variable tipo DATE una fecha que se ha introducido como cadena de caracteres. La cadena de caracteres debe tener el formato:

DD/MM/AA

DTOC

Esta función convierte una variable de tipo DATE a variable de caracteres. CTOD y DTOC son funciones complementarias.

MONTH

Nos da un número que representa el mes en una variable del tipo DATE.

DAY

Es similar a MONTH, pero devuelve un número que representa el día en la variable tipo DATE.

YEAR

Similar a MONTH y DAY, pero devuelve un número que representa el año de una variable del tipo DATE.

DOW

Esta función devuelve un número entre 1 y 7 que representa el día de la semana. Domingo es 1, Lunes es 2, etc.

CDOW

Esta función nos da el nombre del día de la semana de la fecha representada por una variable tipo DATE.

CMONTH

Esta función nos da el nombre del mes representado por la fecha de una variable de tipo DATE.

Comandos alterados

@...GET

Se puede poner ahora un rango para los datos:

@ 10,10 GET TOTAL RANGE 25,100
nos asegura que TOTAL estará en un rango entre 25 y 100. Si el valor introducido no está en el rango especificado, el cursor no se puede mover al siguiente campo.

@...CLEAR

Con este comando se puede borrar la pantalla desde el punto marcado hasta el final:

@ 10,10 CLEAR

APPEND

Usando PgUp o ^R se mueve el apuntador de registros al registro anterior en la base de datos.

BROWSE

El comando BROWSE ha sido implementado de varias formas:

1) Las columnas de datos se pueden bloquear en la parte izquierda de la pantalla, de forma que durante el desplazamiento lateral, permanecen quietos;

2) Ahora hay disponible una función interna GOTO que usa el número de registro;

1) Se ha añadido una opción FIND para índices.

CHANGE

Un comando de pantalla completa como el EDIT, pero se pueden seleccionar campos y/o registros específicos.

CLEAR

Limpia la pantalla.

CLEAR ALL

Cierra todos los ficheros abiertos y libera todas las variables en memoria.

CLEAR GETS

Libera todas las variables direccionadas por GET desde el último READ ejecutado.

CLEAR MEMORY

Borra todas las variables en memoria.

DISPLAY

Se incluyen las cabeceras sobre los campos mostrados. Se pueden eliminar estas cabeceras mediante el comando:

SET HEADINGS OFF

DISPLAY MEMORY
Muestra todas las variables con una representación de su valor interno.

DIR o DIRECTORY
Es el mismo comando que DISPLAY FILES en dBaseII, pero no se usa la parte ON del comando y la palabra LIKE es opcional.

DO
El comando DO permite pasar parámetros. Los ficheros de comandos pueden pasar parámetros a un fichero de comandos de nivel inferior, por medio de la opción WITH. Los parámetros pasados se identifican con el comando PARAMETERS.

ERASE
Hace lo mismo que DELETE FILE en dBaseII.

GOTO
Ahora se puede usar una expresión numérica.

HELP
Esta opción ha sido cambiada radicalmente. Ahora se ha estructurado en forma de árbol con series de pantallas accedidas por medio de menús, o por nombre de pantalla o nombre de materia.

MODIFY COMMAND
Esta opción se parece ahora más a un procesador de textos, y ya no tiene la limitación de 79 caracteres por línea.

REPORT
Se ha expandido la opción TO de este comando para permitir almacenar el texto de un informe en un fichero, para su posterior edición o impresión.

MODIFY REPORT
Es un diseñador de informes en formato de pantalla completa.

RETURN TO MASTER
Devuelve control al fichero de comandos de más alto nivel.

SET ESCAPE ON
Hace que aparezca el mensaje:

¿Abandonar programa? (S/N)

si se pulsa la tecla ESC durante la ejecución de un fichero de comandos. Si la respuesta es S, se aborta la ejecución del comando actual y continúa en el siguiente comando.

WAIT

Se puede añadir ahora un mensaje opcional, como en ACCEPT y en INPUT.

Funciones que han sido renombradas en dBaseIII

@ búsqueda de subcadena. es ahora		AT()
$("ABCDE",2,3)	es ahora	SUBSTR()
!("abcdef")	es ahora	UPPER()
RANK es ahora	es ahora	ASC()
REMARK	es ahora parte de	? y @..SAY
QUIT TO	reemplazada por	RUN
SET COLON	reemplazada por	SET DELIMITER
SET EJECT	incluida en	REPORT
SET HEADING	incluida en	REPORT
SET RAW	incorporada en	TRIM()
TEST	reemplazada por	TYPE()

Funciones que han sido eliminadas

RESET

SET DATE TO

Este es el resumen de los principales cambios en los comandos que han tenido lugar entre el dBaseII y el dBaseIII. En este resumen no se intenta explicar todas las implicaciones que han tenido estos cambios para el diseño de programas y sistemas, los ejemplos del manual lo ilustran convenientemente.

El programa dConvert, suministrado con el sistema dBaseIII realizará algunos de los cambios requeridos para permitir que los ficheros de comandos escritos en dBaseII puedan funcionar bajo dBaseIII.

Si es un usuario experimentado de dBase, deberá leer cuidadosamente el libro dBASE BRIDGE; si no lo es, debería comenzar a usarlo con la opción ASSIST en uso. Para usar dBaseIII en este modo, solo tiene que teclear ASSIST. La opción ASSIST le proporcionará un menú con opciones para:

Crear una base de datos.

Usar una base de datos existente para:

- añadir registros
- cambiar registros
- clasificar en indexar registros
- mostrar y operar con los datos
- borrar registros
- copiar datos seleccionados
- posicionarse en registros seleccionados
- preparar informes

Operaciones estándar con ficheros:

- renombrar un fichero
- copiar un fichero
- borrar un ficheros

Parámetros seleccionados:

Los parámetros de una operación determinada se seleccionan desde un menú, posicionando el cursor en la elección requerida y pulsando RETURN.

INDICE

Comience aquí a leer la otra mitad de la historia sobre:

DATABASE SOFTWARE:

dBase II y dBase III son sistemas de gestión de bases de datos relacionales producidos por Ashton-Tate. Son ampliamente usados en aplicaciones de oficina en casi todos los ordenadores personales más populares.

Pero, la potencia y complejidad de estos potentes sistemas hacen que la mayoría de los usuarios nunca lleguen a dominar más que unos pocos comandos, o quizás usen aplicaciones escritas por otros.

Si usted es el tipo de persona que no se satisface con conocer la mitad de la historia y cree que puede entender lo que hace su ordenador si alguien se lo explica, entonces este es el libro que necesita. He aquí algunas de sus importantes facilidades:

- 100% dedicado a aplicaciones de negocios reales.
- Explicaciones claras de lo que es una base de datos, y de cómo dBaseII/III simplifica este modelo.
- Ejercicios de teclado simples, pero graduados, tanto para el principiante como para el usuario más experto.
- Tratado completo del nuevo sistema dBase III, con referencias a su predecesor dBase II.
- Demostraciones de la conversión de programas del dBase II al dBase III.
- Explicación de los programas de formateo de pantallas de 8 y 16 bits.
- Listado de programas grandes, completos y funcionando.

Sobre el autor

Dr. George Burns es catedrático de la Scottish School of Non-Destructive Testing, que forma parte del Paisley College of Technology. Tiene una amplia experiencia en la enseñanza de muchos tipos de programas para los negocios.

ISBN 84 - 86381 - 24 - X

RA — MA
Carretera de Canillas, 144.
28043 Madrid.

dBase III y dBase II

George Burns

ra-ma

AMSTRAD CPC



MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA



https://acpc.me/